开源世界旅行手册

wizardforcel

Published
with GitBook



目錄

开源世界旅行手册

目录

范例清单

授权

致谢

本书以《ubuntu教程》为基础进行扩充，可以看作是《ubuntu教程》的第二版

一些章节，我会在 linuxtoy.org 网站上发布初稿，根据网友的意见进行修改。感谢 `linuxtoy.org` 站长 **`Toy`** 的大力支持

第 31 章 完美工作站 *Archlinux*，根据《打造完美的 Linux 桌面——Archlinux》改写。所用图片,由 `Toy` 友情提供

第 29 章 *ConTeXt* 入门指南中关于 ConTeXt 的内容，取材于《ConTeXt MkIV 学习笔记》，在神秘人士的指导下完成

关于 screen 的内容，由 `roylez` 担当顾问

`mdz` `hxl` `Reiase` `niuniu` `qxzw` `木子雨木木` `小连猪` 等朋友参与了本书的审校

作者的邮箱地址为 2999am@gmail.com ，无论有任何建议，或者希望参与这本书的写作，请与作者联系

序言

目录

千言万语，从何说起：

改变

在最初的提纲中，现在的四个部分依次为：杂谈、教程、解决方案、历史

只是我希望，在技术文档冰冷的线条上，涂抹些温暖的色调

当然，这个文档目前还是一些冰冷的线条，但是我会一直努力

最早的版本发布计划为：预览版、正式版、修订版

因为《Ubuntu教程》的经验告诉我，第一次公开发布的版本流传最广。所以我对于发布第一个版本，总有一些忌惮

现在，我决定使用浮点数作为版本号

里程碑

因为我自己，把这次发布作为一个里程碑

不是由于故地重游的感极而喟，不是由于风雨兼程的劳碌困顿，甚至不是由于，孕育的艰辛和初生的喜悦

而是由于，这个文档的写作方式由闭门造车，变为海纳百川

我希望，以此文档为基础，在志愿者的努力下，打造一份中文原生的开源教科书

我希望，它能够为前行者铺就一条简陋的石子小径

我希望，它是天涯的号角，水手们，我们的船即将启航

我希望，它是海角的远眺，波涛啊，世界的尽头在哪里

我希望，它是暴风雨中的闪电，刹那的光芒，请为我照亮朦胧的远方

我希望，它是回归的候鸟悠远的长啸，春风啊，请为我扬起这面炽热的旗帜

某人 于 某日 29:99 AM

更新纪录

2009-04-12

整理

2009-02-11

修正一些错误（BY:小连猪）

2008-12-20

完成:UNIX 缩写风格

完成:awesome

2008-12-13

调整:zsh 配置文件

2008-12-12

增加:完全用 Gnu/Linux 工作

增加:Unix

增加:linux

增加:Gnome与KDE的战争

修正:Grub

修正:Latex

2008-12-08

发布: `0.1版`

2008-12-05

修订:git

修订:archlinux

2008-12-05

增加:lftp

2008-12-1

完成:Archlinux

调整:在英文单词和中文之间增加空格

2008-11-31

计划:Archlinux

计划:XeTeX

2008-11-30

增加:授权

2008-11-29

修正:git 版本库索引 => 索引

2008-11-25

完成:git

2008-11-19

完成:conTeXt 配置

增加:《无根的根》 导言

2008-11-17

完成:服务器

2008-11-15

完成:正则表达式

2008-11-09

发布: `内部交流版`

导读

Linux并不意味着不苟言笑的命令行，我多数时间在图形界面中工作

尽管如此，本书仍然不会尝试指导读者，如何点击某个菜单，如何启动某个程序……

同样，如非必要，本书也不会为大多数操作配备截图

本书分为四个部分：气候、地理、景观、地质

气候篇中的内容为课外读物，包括一些杂谈随笔，可以增长知识

地理篇为必修课，它是这本书的核心，包含一些基本教程。这一部分的内容，建议熟读

景观篇为选修课，里面的内容为实用的解决方案，但并不是每个人都需要

地质篇的内容为开源运动史

第 17 章 核心工具集 可以作为 "命令手册"

如何写作科技文档

科技文档相对特殊，把它写得通俗易懂而又不失简洁，几乎是一件不可能完成的任务。

尽管如此，通过有意识的使用一些技巧，还是能够尽可能的接近这个目标。 我这里有一些个人的经验，请看下面的三种表述:

- 第一个例子[1]

```
就像前面提到的，所有USE标记都声明在USE变量里面。
为了让用户能方便地查找和选择USE标记，我们提供了一份默认的USE设定。
这些设定是我们觉得Gentoo用户通常都要用到的USE标记的集合。
这个默认设置在make.defaults文件——你的profile的一部分——里声明。

你的系统使用的profile是符号链接/etc/make.profile所指向的目录。
每个profile叠加于某个更大的profile之上，最终的结果是这些profile的并集。
初始profile是base profile（/usr/portage/profiles/base）。
```

- 第二个例子

```
/etc/make.profile/目录是一个符号链接，里面包含一些make.defaults文件，实际指向的是以下文件，
可以在这些文件中设置USE标记(不可更改)
```

- 第三个例子

```
/etc/make.profile/目录是一个符号链接，里面包含一些make.defaults文件，放置开发者设置的USE标记：
```

目的明确. 手把手的操作步骤与宏观的概念描述、面向新手与面向老手，肯定会有很大的不同。要根据需要决定你的方向

第一个例子是给忠实用户使用的权威官方文档，后面例子中是向新手介绍的简要说明，目的不一样，所以介绍的详细程度不同

简明清晰. 在有些时候，要改变一下思路，从另一个角度叙述(在翻译文档时，这种技巧比较有用)。

第二个例子是最常见的介绍方式，它暴露了过于复杂的细节：实际指向、不可更改，而没有说明关键细节：开发者设置，还有一些冗余信息：可以在这些文件中设置

第三个例子中，在介绍了USE的概念之后，说明这些USE是开发者设置的，就可以暗示很多事情。如果说“系统默认、不可更改……”你最好说明为什么，因为“不可更改”这种声音，对于Linux用户来说无疑有些刺耳……

控制复杂度. 描述的复杂度必须控制，不要假定用户对Linux很熟悉……如果所有的文档都使用这样的假定，那么大多数用户很难熟悉Linux

在分析了自己的目的之后，你要有一个取舍。你要确定读者是否需要明白什么是“profile”

针对“profile”，如果不深入解释，有点虎头蛇尾，而科技文档的一个很重要的特点是“完整”；如果解释，则增加了一个不必要的分支

不要过度隐喻. 对于软件，只要不是在源代码的层面上描述，都算作隐喻。只是有的隐喻不明显，如启动、运行(它们的喻体是符号化的“系统”)；而有些隐喻则比较形象，例如电视机、遥控器

如果不是难以描述的概念，不要用隐喻

如果确实需要使用形象的隐喻，特别是在短时间内多次使用，请确保喻体的相关性。电视机:系统、遥控器:终端 这样的比喻容易理解，而电视机:系统、手机:终端 的隐喻，多少令人费解

限制术语的使用. 术语本身拥有需要解释的特点。 像第一个例子中的“并集”，这是集合中的术语，并不是每个读者都熟悉集合的概念；而“并集”指多个集合相加，熟悉集合概念的读者，可能会深究被加在一起的集合……如果不引入这个术语也可以解释清楚，那么不要引入它

基础概念. 不要认为介绍基础概念是吃力不讨好的工作，尽管读者可以从操作步骤中得出基础概念

既然大多数的描述都是隐喻，在描述基础概念的时候，不要顾忌使用隐喻。读者得到一个基本概念之后，可以更容易的理解接下来的描述。(“解释学之弧”)

雕琢实例. 不要试图通过一个实例阐释许多个概念，不要随手抓一个什么东西当作实例。应该构建最简化的实例，每个实例只讲一件事(K&R 《The C Programming Language》)

充分利用文档工具的特性. 脚注，calloutlist

多数文档工具都可以使用脚注，使用脚注作简短的说明，可以确保在不增加分支的情况下，对概念作必要的说明

calloutlist用来作密集注释。虽然你可以把这部分内容分成多段描述，但是使用calloutlist，可以清晰的列出主干，更容易阅读[2]

厚积薄发. 控制表达的冲动。很多文档作者都容易犯这样的错误，旁征博引、滔滔不绝……(《鸟哥私房菜》)可能你知道的确实挺详细，但请记住，你不是自说自话，而是写给读者。请考虑一下读者希望看到什么内容，而不是你想写什么内容

---

[1] 本例来自gentoo中文手册

[2] 另一方面，只有 DocBook 支持 calloutlist，这意味着，如果从 DocBook 转换到其它格式，calloutlist 将会制造一些麻烦 在代码环境中使用的注释，很大程度上可以替代calloutlist的作用

部分 I. 气候

目录

第 1 章 GUI? CLI?

目录

定义

`GUI`，Graphical User Interface，图形用户界面。用户界面的所有元素图形化，主要使用鼠标作为输入工具，点击图标执行程序，使用按钮、菜单、对话框等进行交互，追求易用，看起来比较美

`CLI`，Command Line Interface，命令行界面。用户界面字符化，使用键盘作为输入工具，输入命令、选项、参数执行程序，追求高效，看起来比较酷

CLI 的优点

对于 Linux ，命令行不是必需的，点点鼠标，同样可以完成所有的事

而 GUI 的易用，使得一个从没接触过 Linux 的初学者，也可以通过点击鼠标作一些事情，比如他可以点击文件夹，启动文件管理器，在文件上点击右键，通过菜单对文件进行一些操作

这是不是意味着，CLI 就应该放到博物馆里供人凭吊？

当然不是这样的，实际上，CLI 在熟练用户中仍然大行其道，因为它的效率高

回忆一下，在 Windows 系统中通过 GUI 启动“计算器”：开始→所有程序→附件→计算器，其实这不算什么，只不过点了几次鼠标而已……

在这个计算器中，用鼠标点击上面的按键输入表达式，或者某些程序要求你输入密码的时候，弹出一个小键盘，要求你点击上面的数字……不得不说，这种设置十分的人性化，计算器看起来跟真的很像，到银行取钱，也用用于输入密码的小键盘……不过问题的关键在于，这种图形界面模拟的键盘是用手来按的，而不是用鼠标点的

既然如此，为什么不直接按键盘？

在 CLI 下使用计算器，只要输入 `bc` 就可以启动计算器，输入一个复杂的公式 `1+1` ，回车， 得到结果 2

好了，我必须承认，这个例子有点过分，因为算术不是计算机的主要用途

第 2 章 UNIX 缩写风格

目录

缩写习惯

无聊和乏味的工作是罪恶

-- Eric S. Raymond

构建于图形界面之上的操作系统，使用鼠标作为主输入设备， 是否使用缩写并不重要。比如 Windows 系统中的目录，几乎都是全称…… 点击两次鼠标进入文件夹 `pf` ， 并不意味着点击13次才能进入文件夹 `Program Files`

而构建于命令行之上的操作系统，如 Linux，只要3个字母以上的单词，几乎都要缩写。 例如： **cd** 命令是 `Change Directory` 的缩写。 作为常用命令，如果使用它的全称 `Change Directory` ，绝对是无聊和乏味的工作。

最常见的缩写，取每个单词的首字母，如

cd	Change Directory
dd	Disk Dump
df	Disk Free
du	Disk Usage
pwd	Print Working Directory
ps	Processes Status
PS	Prompt Strings
su	Substitute User
rc	Run Command
Tcl	Tool Command Language
cups	Common Unix Printing System
apt	Advanced Packaging Tool
bg	BackGround
ping	Packet InterNet Grouper

如果首字母后为“h”，通常保留

chsh	CHange SHell
chmod	CHange MODe
chown	CHange OWNer
chgrp	CHange GRouP
bash	Bourne Again SHell
zsh	Z SHell
ksh	Korn SHell
ssh	Secure SHell

递归缩写[3]也属于这一类，如：

GNU	GNU's Not Unix
PHP	PHP: Hypertext Preprocessor
RPM	RPM Package Manager
WINE	WINE Is Not an Emulator
PNG	PNG's Not GIF
nano	Nano's ANOther editor

有些缩写可能有多种定义，如：

```
`rpm`
RPM Package Manager
RedHat Package Manager

`bc`
Basic Calculator
Better Calculator
```

这方面 Emacs 可谓独领风骚：

```
`Emacs`
Editor MACroS
Emacs Makes A Computer Slow
Escape Meta Alt Control Shift
Emacs Makers Are Crazy Sickos
Emacs Makes All Computing Simple
Emacs Makefiles Annihilate C-Shells
Emacs Manuals Always Cause Senility
Emacs May Allow Customized Screwups
Emacs Manuals Are Cryptic and Surreal
Eventually Munches All Computer Storage
Eight Megabytes And Constantly Swapping
Elsewhere Maybe All Commands are Simple
Excellent Manuals Are Clearly Suppressed
Emacs May Alienate Clients and Supporters
Except by Middle Aged Computer Scientists
Extended Macros Are Considered Superfluous
Every Mode Accelerates Creation of Software
Each Manual's Audience is Completely Stupefied
Exceptionally Mediocre Algorithm for Computer Scientists
Easily Maintained with the Assistance of Chemical Solutions
Eradication of Memory Accomplished with Complete Simplicity
```

如果只有一个单词，通常取每个音节的首字母：

cp	CoPy
ln	LiNk
ls	LiSt
mv	MoVe
rm	ReMove

对于目录，通常使用前几个字母作为缩写：

bin	BINaries
dev	DEVices
etc	ETCetera
lib	LIBrary
var	VARiable
proc	PROCesses
sbin	Superuser BINaries
tmp	TeMPorary
usr	Unix Shared Resources

这种缩写的其它情况

diff	DIFFerences
cal	CALendar
cat	CATenate
ed	EDitor
exec	EXECute
tab	TABle
regexp	REGular EXPression

如果某种缩写比较深入人心，例如“mesg”代表“message”，在新的复合缩写中，将沿用这种缩写方式

dmesg	Diagnostic MESsaGe
sed	Stream EDitor
stty	Set TTY
fstab	FileSystem TABle
passwd	PASSWorD

有些缩写中，第一个字母“g”，代表“GNU”

awk	Aho Weiberger and Kernighan
gawk	GNU AWK
gpg	GNU Privacy Guard
grep	GNU Regular Expression Print
egrep	Extended GREP

> [3] 定义中包含自身缩写，如 GNU：

`GNU` 's Not Unix

使用这个定义来解释定义中的缩写：

(GNU's Not Unix)'s Not Unix

这意味着它是可以无限递归的：

(((((GNU's Not Unix)'s Not Unix)'s Not Unix)'s Not Unix)'s Not Unix)'s Not Unix ……

命令选项，从a到z

Linux 命令的选项繁复庞杂，让人眼花缭乱。不过这些选项往往具有相对固定的涵义，熟悉了它们，记忆便不再困难

-a

all : 全部，所有 (ls , lsattr , uname)

archive : 存档 (cp , rsync)

append : 附加 (tar -A , 7z)

-b

blocksize : 块大小，带参数 (du , df)

batch : 批处理模式 (交互模式的程序通常拥有此选项，如 top -b)

-c

commands : 执行命令，带参数 (bash , ksh , python)

create : 创建 (tar)

-d

debug : 调试

delete : 删除

directory : 目录 (ls)

-e

execute : 执行，带参数 (xterm , perl)

edit : 编辑

exclude : 排除

-f

force : 强制，不经确认(cp , rm ,mv)

file : 文件，带参数 (tar)

configuration file : 指定配置文件(有些守护进程拥有此选项，如 ssh , lighttpd)

-g

-h

--help : 帮助

human readable : 人性化显示(ls , du , df)

headers : 头部

-i

interactive : 交互模式，提示(rm , mv)

include : 包含

-k

keep : 保留

kill

-l

long listing format : 长格式(ls)

list : 列表

load : 读取 (gcc , emacs)

-m

message : 消息 (cvs)

manual : 手册 (whereis)

create home : 创建 home 目录 (usermod , useradd)

-n

number : 行号、编号 (cat , head , tail , pstree , lspci)

no : (useradd , make)

-o

output : 输出 (cc , sort)

options : 选项 (mount)

-p

port : 端口，带参数 (很多网络工具拥有此选项，如 ssh , lftp)

protocol : 协议，带参数

passwd : 密码，带参数

-q

quiet : 静默

-r

reverse : 反转

recursive : 递归 (cp , rm , chmod -R)

-s

silent : 安静

size : 大小，带参数

subject

-t

tag

type : 类型 (mount)

-u

user : 用户名、UID，带参数

-v

verbose : 冗长

version : 版本

-w

width : 宽度

warning : 警告

-x

exclude : 排除 (tar , zip)

-y

yes

-z

zip : 启用压缩 (bzip , tar , zcat , zip , cvs)

第 3 章 版本号的迷雾

目录

有时我会听到这种说法：我安装了 9.0 版的 Linux，然后……

目前最新的内核为 2.6.27 版，何来 9.0 版之说？根据经验，我大概可以揣测出，9.0 为某一发行版的版本

需要指出的是，Linux 并不是一个完整的系统，它只是内核。没有内核系统不能运行，什么都干不了；但只有内核还是什么都干不了，一个完整的系统，是包含内核在内的一系列软件工具包

开源运动由许多独立的软件项目构成，如果最终用户需要自行获取这些软件包，然后把它们组装起来，成为可以运行的系统，就会浪费很多时间；而且组装一个系统并不是一件容易的事情，恐怕大多数用户无法作到。

于是出现了一些组织或个人，将内核和其它软件组装在一起，作为一个完整的系统发布，这就是发行版。安装大多数发行版，如 ubuntu、archlinux，就是通过安装程序，将已经组装起来的系统安装到计算机上

发行版

各种发行版都有自己的版本命名方式，有些通过发布时间来命名，像 archlinux：

```
archlinux 2008.06
```

表示2008年6月发布该版本

很多发行版除了版本号，还有发布代号。比如 ubuntu，选择一种动物作为吉祥物^_^!!

```
7.04 - Feisty Fawn
7.10 - Gutsy Gibbon
8.04 - Hardy Heron
8.10 - Intrepid Ibex
9.04 - Jaunty Jackalope
```

> ❶ 发布时间为 08 年 10 月
>
> ❷ 无畏的北山羊

ubuntu 每6个月发布一次新版，所以比较隆重一点；而 archlinux 是滚动更新，发布新版只是便于新用户安装

一些历史悠久的发行版，倾向使用序数作版本号，例如：

```
Fedora Core 10
openSUSE 11
Debian 5.0
FreeBSD 8.0
```

- 严格的说，FreeBSD 并不是 Linux 发行版，因为它使用的不是 Linux 内核

内核

Linux 内核的开发，在两个分支上同时进行，稳定分支和实验分支。稳定分支相当健壮，可用于生产环境；而实验分支中包含一些新的特性，还不够成熟；待实验分支的代码经过充分测试，被证明足够成熟，便会被转移到稳定分支

这种开发模式既保证了有一个可靠的稳定版用于生产，又保证了能够大胆的在内核中应用新技术，大部分开源项目都使用此模式开发(包括 FreeBSD 等)

来看看内核版本号

```
2.6.27-2-i686
```

> ❶ 主版本号。革命性改进，这个版本号在几年内应该不会升级
>
> ❷ 次版本号。重大改进，偶数为稳定分支，奇数为实验分支
>
> ❸ 修正版本号。重大修正
>
> ❹ 修补版本号。一些 BUGS 的修补
>
> ❺ 目标架构。i686 表示 intel奔腾Ⅱ或以上级别 CPU

第 4 章 Vim 还是 Emacs

第 5 章 DocBook 还是 TeX

第 6 章 完全用 Gnu/Linux 工作

目录

作者:王垠

> 注意：本章脚注为编者添加

我已经半年没有使用 Windows 的方式工作了。Linux 高效的完成了我所有的工作。

GNU/Linux 不是每个人都想用的。如果你只需要处理一般的事务，打游戏，那么你不需要了解下面这些了。

我不是一个狂热的自由软件份子，虽然我很喜欢自由软件。这篇文章也不是用来推行自由软件运动的，虽然我觉得自由软件运动是非常好的。

这篇文章也不是用来比较 Linux 和 Windows 内核效率，文件系统，网络服务的。我现在是作为一个用户而不是一个开发者来说话的，我们的讨论是基于操作，应用层面的。是为了告诉大学里还不了解，或者不理解 UNIX 的科学工作者和大学生，UNIX 比 Windows 更适合用于科学研究工作，请大家理解 UNIX 的工作方式，不要用 Windows 的标准来要求 Linux，而要用一个科学工作者的标准来要求自己，用 UNIX 的思想来武装自己。

我显然是反对在大学，特别是理工科专业推广 Windows 的。我也反对在对“娃娃”们的计算机启蒙教育中使用 Windows。因为 Windows 不论从技术上，经济上，思想风格上都是与我们培养高科技人才的目标格格不入的。Windows 的流行属于历史遗留问题，爷爷一级的人当然已经不可救药[4]，但是我们不应该让下一代继续走上歧途。

> [4] 其实爷爷一级的人也不是不可救药

UNIX 不是计算机专家的专利

当我建议一些非计算机专业的人用 Linux 的时候，很多人说：“UNIX 是计算机系的人用的，我们不能理解。” “UNIX 是男孩用的，我们女孩不用。”

但是其实世界上的大多数科学家和工程师几乎用的都是 UNIX 作为他们的电脑工具。就因为它简单，可靠，稳定，强大，有趣。甚至很多时候 UNIX 就是唯一的选择。

你说：“我们都会用 UNIX 的话，你们计算机专业的人还用来干什么？” 很荣幸的告诉你，计算机专业的有一部分人就是专门为你们提供这样强大而方便的计算机工具的。如果他们制造的工具只有自己会用的话，那这个工具还有什么用？

理解 GNU/Linux

不要用 Windows 的标准来要求 Linux。

由于GNU/Linux这个词太长，下面如果没有特别指明，“Linux”就是指“GNU/Linux”。

在这个年代，恐怕没有人需要我来介绍 Linux 是什么了吧？如果你觉得“Linux 只不过是跟 DOS 差不多的东西”，那请问问你旁边的 Linux 用户，Linux 到底是什么？

那为什么我还要写一篇这样的文章？因为，我发现还有很多人不不理解 Linux 和 UNIX，虽然他们也在用它，但是他们有时会问：“为什么 Linux 不能像 Windows 那样 ……？”，“怎么 Redhat Linux不能 mount NTFS 分区！”，“Linux 下用什么整理硬盘？”，“什么时候 OpenOffice才能完全兼容Word文件啊？”，“现在还有什么Windows能干的事情Linux干不了的？”……

他们有40G的硬盘，却只为 Linux 分配了2G空间，有时还抱怨“这个东西怎么占这么多硬盘！”似乎 Windows 该占用大部分硬盘。他们把重要的数据装在Windows的分区，似乎信不过 Linux。他们总是到处寻找新奇的，好看的GUI程序，对命令行的东西一概不屑一顾。他们对 Drag&Drop，菜单配置，自动升级非常感兴趣。他们如果找到一个很像 Windows 程序的 Linux 程序，一定会很高兴的说：“哈哈！Linux 也能……了！”如果Linux在某种测试中胜过 Windows，他们会高兴得跳起来。他们没有办法用Linux解决问题的时候，甚至用Wine[5]来运行Windows程序。有时实在没办法，只好重起到 Windows，或者干脆省得麻烦，在 Windows 下装一个 VMWare[6] 虚拟一个 Linux 玩。

你如果出现了上面的情况，说明你的思想受到了 Windows 的某种潜移默化的影响和误导。你没有能够从本质上理解存在于 Linux 身上的 UNIX 思想。你支持 Linux，你喜欢 Linux，你能从中感觉到快乐，这非常好。你现在只需要明白的是：Linux 从来就不是一个玩具，它是天才UNIX的后代。UNIX 是自晶体管发明以来最伟大的发明[7]，它从诞生那一天开始就比 Windows 的设计出色。

你要体会什么叫做“设计”，一个糟糕的设计并不是到后来缝缝补补就可以变好的，而一个出色的设计，不但可以以不变应万变，而且可以影响到后来者。一个出色的设计配上一个出色的实现，那就是非常出色的发明。Linux 就是这样的一个出色的发明。 Linux 并不需要追赶 Windows，也不需要打垮微软。它的最终目标是改变整个计算机世界，还人们自由，给人们乐趣和方便。

Unix 是简单的，你不需要成为一个天才也能理解这种简单。

UNIX的设计者 Dennis Ritchie 说：“Unix is simple. It just takes a genius to understand its simplicity.” 但是我不这么认为，因为我不是一个天才，但是我却勇敢的把 Windows 完全删除掉，遇到不明白的事情的时候努力用 UNIX 的方式去解决，而不是寻求 Windows 的帮助。现在我体会到了 UNIX 的思想和好处，我可以用比 Windows 高效几倍的效率工作。因为我相信这样的信念：“Windows 能办到的事 Linux 一定能办到，而且办的更好。”

这小节开头的话应该改成：“Unix 是简单的，你不需要成为一个天才或是计算机专家。但是在这个冲斥着 Windows 错误观念的世界，你需要信念和勇气才能理解它的简单。” 我下面就告诉你一些我理解到的东西。首先，你要知道的是微软在国际科学领域是根本没有地位的。

> [5] Wine 现在已经非常实用了
>
> [6] 其实这没什么不好
>
> [7] 自图灵奖诞生以来，其获得者一直都是计算机领域的科学家与学者，而在所有这些届的图灵奖中只有唯一的一届有个例外，那就是Ken Thompson与Dennis M. Ritchie，他们都是计算机软件工程师。

由于Unix与C语言的深远影响，1983年美国计算机协会将当年的图灵奖破例颁给了作为软件工程师的Ken与Dennis，并在当年还决定新设立一个奖项——软件系统奖，以奖励那些优秀的软件开发者，当然首个软件系统奖也是非他们两人莫属了。

微软的地位

微软的名声在欧洲和美国的大学里，特别是在计算机系里之坏，大家可能有所耳闻。我认识的 MIT，Stanford 的教授，贝尔实验室的专家，甚至一个欧洲小国的高中计算机老师都绝口不提微软的名字。在他们眼里，微软只是一个没有真技术，专靠在落后国家商业宣传和垄断经营的小公司。这个“小”并不是说它人少，钱少，而是说它先进技术少。

我上次和王益合作写了一个算法演示程序，那个算法是贝尔实验室一位科学家Steven Fortune很天才的发明，为了程序能够被身边大多数人使用，我们选择了 VC+MFC 作为平台。我在分析算法时还得到 Fortune 很热情的鼓励，寄给我一份资料，还多次回信耐心的给我讲解了很多细节。但是程序完成之后，我把样品发给 Fortune，他回信说：“对不起。我机器上没有 MFC。” 话说的很客气，但是我已经感觉到了他对 Windows 的不屑。然后我把 MFC 静态编译进程序再发给他，他就没有再回信了。他显然不是瞧不起我，而是确实有难处。

你能感觉到这位科学家对微软和 Windows 是什么态度了吧？不是反感，而是他心里根本没有 Windows 这个东西！微软在高科技领域没有发展，那么它怎么生存呢？到发展中国家去发展一下，他们的人民还对电脑一无所知，我说不定甚至可以打入大学的计算机系呢。我送他们软件，我捐钱盖大楼，我出钱找图灵奖获得者来演讲，让他们觉得我们都是科学家！

好了，现在全国的大学包括清华，几乎所有人机器必装盗版 Win2000，Office XP，学校的选课系统是非IE不能正确浏览，论文用 Word 编辑，演示用ppt做，email 的通知附件是 doc 文件，你不用 Word 打不开，连 863 项目都用 VC 写程序了。我很久以前就看到一份报纸说，“微软为什么不严厉打击盗版？” 这篇文章说，微软非但不打击中国的盗版行为，而且有放任之趋势。放长线吊大鱼，“以后我要你们加倍的来还我！” 确实如此，它的目的快实现了。[8]

Windows 笼罩下的中国计算机教育

说句丢脸的话，比尔盖茨很久以前是我的偶像……

在中国，比尔盖茨被很多人奉为神圣，“少年电脑天才”，甚至有的人提到他的名字就做出“抱拳对天”的姿势。很多人谈到微软的“新技术”，“高科技” 都是眉飞色舞。各种“VC编程圣经”，“深入了解 Visual C++”之类的书，在开头几页都会出现非常肉麻的字眼，“在那团团的混沌中，一个开天辟地的精灵，Windows 1.0，诞生了……”

微软的软件被这么多人盗用，那么人们是怎样使用这些盗版程序的呢？先看看电脑培训班，教的都是一些 DOS 命令，打字，Windows 基本操作，Word 文档处理，PowerPoint，高级班可能有 Excel，Access…… 参加各种微软认证考试，MCSE，MSDE 的人络绎不绝。考试辅导班都贴出了“280元，考过为止”之类的字样。考试参考资料更是昂贵，有些电脑书店整整两书架都是“Microsoft Press”的东西。我有个同学参加认证考试，每门考试都要200多元。而且你一次考不过可以再考，又要交钱。他后来还津津乐道跟我说，看我，花了XXXX(一个四位数)元考过了微软认证，得到一张比尔盖茨亲笔签名的证书和价值6000元的 Windows XP 内部发行版。

“电脑要从娃娃抓起”，我们再来看看娃娃们学的是什么。大部分家长给孩子买了电脑之后，他们首先就会装一个盗版的 Windows，然后买来盗版的游戏开始玩。如果哪个孩子会用 Delphi 编程序，那可不得了。报社记者，电视台争相报导，说，某某学校的初中生某某，在别人都还在玩电脑游戏这种“初级阶段”的时候就已经用 Delphi 写程序了。镜头还瞄准了他显示器上面的像框中的比尔盖茨头像！

我刚进入大学计算机系时还不懂得什么是操作系统，因为我以前只用过“中华学习机”。看到新入学的同学们各个谈论的都是 “Windows 95”，“VC”…… 我简直觉得我落后了好几十年一样，整个一土人，根本跟他们答不上话。好不容易找到一个比较熟的同学问了一下：“你们天天谈论的瘟95是什么啊？”答：“win95就是一个操作系统，跟DOS是一类。”“朵死是什么？” “你连DOS都不知道是什么？别在计算机系混了。” 学校上课当然不讲VC编程之类的东西，但是上 Pascal 的老师有一次就说：“嗨，我们学校真是落后。现在别人都用 C, C++，甚至 VC 了，我们还在讲 Pascal。不知道什么时候才能有VC课啊。你们出去也是要用VC的，只好自学了。” 于是，有些同学很多时候上课都捧着一本很重的“Windows 编程大全”之类的书，根本没有听课。吃饭时就念念有词的跟我说，“代码的优化是无止境的”，“匈牙利命名法真是伟大的发明 ” …… 这就是中国很多大学计算机系的情况。

感觉到无知了？这不是偶然的，而是微软长久以来埋下的伏笔。它要让无知的大家都把它奉为神圣，它要让支持UNIX，Xwindow的人一旦说 UNIX 好，Xwindow 好的时候，都被一群人围着说教：“这个 Windows 也能做到 ”，“你对 Windows 有偏见”，“微软才是主流啊”，“你敢瞧不起 win2k？”，“.NET 就是世界潮流”，“微软的毕竟是新技术”，“有钱就是有技术”…… 甚至在一番论战比较后败下来还是要说：“Windows 性能差点，但是易用性强”，“Windows 是老百姓用的，要求别那么高 ”，“微软那么有钱，以后想超过 UNIX 还不容易吗？”……

发达国家的计算机教育

我前段时间在 USENET 发文问有关 Scheme 语言的问题时，认识了一位丹麦人。他解决了我所有的问题，并且建议我阅读一些很“深奥”的有关程序语言语法，文法的书，他告诉我很多网站可以学习 LISP，Scheme，人工智能，算法。他叫我看 Jonathan Rees 的论文 "Syntactic Closures"。他还打包给我寄过来一份 MIT 的 "How to Design Programs"。他说他在自己的 PC 机上装的是 Linux，他用 Emacs 编辑，运行 Scheme 程序。他对 Emacs 的了解和爱好真是使人惊讶。他大学本科毕业时做的毕业设计是一个 Scheme 解释器。这对于我来说是望尘末及了。

他是那么的不厌其烦，我的每一个问题他都详细的回答。我有时都觉得过于详细了，怎么这么耐心啊？我觉得他似乎是我的高中老师。他是什么样的人呢？我好奇的打听了他的情况。原来，他是丹麦一所普通高中的计算机老师。

他说他在高中里讲授程序设计和算法，计算机语言文法。他说用 Scheme，他的学生不用再为内存泄漏等程序语言本身的问题而烦恼，而专注于问题和算法本身。有利于培养学生解决问题的能力，特别是用计算机解决数学问题的能力。

天哪！为什么欧洲出现那么多数学家，几何学家？你看看别人重视的是什么！我们的计算机教育如果继续这样下去，只会沿着弯路越走越远！

微软和它的朋友们的如意算盘

下面来看看微软的收入是怎么来的。首先，Windows 98系列操作系统，一个就是 100 多美元，每次升级又是几乎同样的价钱。Windows NT 还要贵几倍，而且有用户数目限制，5个用户的，10个用户的…… 以后如果要增加用户数目还要按比例付钱。

花了如此多钱买来的操作系统就能用了吗？它竟然连压缩程序都没有提供！你装上 Windows 之后一般第一件事就是去下载一个 WinZip 吧，“只要 29 美元”。Windows 会中病毒啊，马上花 70 美元买一个 Norton AntiVirus 吧。还有黑客呢？再买一个 Norton Internet Security 好了，100 美元。系统需要优化，磁盘需要整理，买一个 Norton System Works 是你最佳的解决方案，100美元。

可是你现在还是不能干正事啊！你想要一个 Word, PowerPoint？那就买一套 Office XP 吧，一起买便宜些，$459.90。

那些程序不会用啊！那些菜单怎么设置，到底有什么功能啊？看“帮助”也学不会。买本书看看吧，我推荐 “Special Edition Using Microsoft Office XP”，不贵，$27.99。这本书里面大部分是屏幕抓图，还是买一本旧的比较划算，$17.85。

你如果只是当个秘书，上面的差不多还凑合了。可是你有更高的追求，你想成为 Windows 程序员。首先买一个 Visual Studio.NET 吧，要不然怎么编译程序。$494.95。

为了紧跟微软动向，世界潮流，不能不注册个 MSDN 什么的吧？这个贵一点，不过物有所值啊，$2,799。

嗯，你现在已经是上层阶级，白领人士了。你现在可以像这样“自由”的，“安全”的生活了

> [8] 黑屏事件

为什么要反对使用 Windows

很多人都说不应该完全否定 Window，Windows 也有它的长处。不应该骂微软。

对。Windows 容易操作，适合普通用户。如果微软把它自己定位在 P&G，Philips 那样的地位，能够给我们的百姓提供周到的，完善的，价廉物美的服务。那我肯定是很喜欢它的。

但是从上面的种种情况说明，微软是一个野心极大的国际垄断组织！它的产品没有一个是不出问题的：Windows 不稳定，容易中病毒，而微软不为大家免费提供杀毒软件。我就是要让你们花钱买我的朋友 Symantec 的杀毒软件，谁叫你们已经上了我的贼船？这叫什么售后服务啊！

你买来微软的程序，安装的时候一般都有一个协议，说："由于微软的程序造成你的数据损坏或丢失，微软概不负责。" 我想很多人肯定觉得这个不合理，不想按那个 "I accept"。但是你的软件买都买来了，钱都花了，现在一按 "I decline"，安装程序马上就会退出。你只好被迫点击了 "I accept"！这不是不平等条约吗？

我已经目睹了好几个朋友的文档被 Microsoft Word 损坏，有的是编辑了十多天的30多页的论文，有的是费了很大工夫做出来的个人简历，那个朋友为此失去了到自己向往的 P&G 工作的机会。就在他要投简历的前一个晚上，就在那一瞬间…… 不知道他痛哭的时候有没有想起要投诉微软，可是谁叫我们用的都是盗版呢，况且你还点击了 "I accept"。

微软仗势已经占有大部分PC市场，制定不符合国际标准的"微软的标准"，以不合理的方式压制其它公司的软件，这个问题已经在美国司法部闹了很久了。他甚至在 Windows 系列操作系统中放置能够通过网络泄漏用户信息的代码，以至于 Windows 刚进入澳大利亚时被澳大利亚政府禁止使用。

有些人说："微软毕竟开创了一个历史，造就了今天的 IT 行业。" 但是，如果没有微软，我们今天早就用上非常稳定，非常可靠，非常方便，非常"傻瓜"的软件了！微软是阻挡信息技术发展的罪魁祸首。

微软的程序的工作方式(注意，我只是说操作方式，病毒的事情另外算)确实适合于一般家庭，上上网，发发邮件，打打游戏都不错。可是微软却要把自己包装成什么 "高科技"企业，要在世界各地设置"研究院"，在大学计算机系赠送不适合用于科研的 Windows 产品，甚至出钱请图灵奖得主来中国畅谈"二十一世纪的计算"，还在大会上宣传自己的 .NET 技术。非要把别人认为自己是科学的，自己是领导世界高科技的。但是呢？它什么高科技也没有。欧洲，美国，哪一个关键部门在用微软的东西？NASA? DOE? CERN? 你仔细想一想，微软的程序对人类到底有什么重大作用？

什么是 Windows 能干而 Linux 干不了的事情？

“Windows 能干而 Linux 干不了的事情，那就是不需要干的事情。”

有个朋友看我半年没有用 Windows，有时就会问我：“你只用 Linux，有没有发现有些 Windows 能处理的事情 Linux 干不了？”

我回答说：“Windows 能干而 Linux 干不了的事情，那就是不需要干的事情。”

Windows 能做的有益的事情 **Linux** 都能做

Windows 下的某些功能确实是我们需要的，那么 Linux 的开发者们和用户也需要这种功能，他们就会去实现这种功能，而且比 Windows 的方式好得多。由于大多数科学家，工程师用的都是 Linux 或者某种商业 UNIX, 所以几乎所有商业的科学工程程序，比如 Matlab, Mathematica, AutoCAD, Candence的，Synopsys 的，Avant! 的……全都是先有 UNIX 的版本(包括Linux)，然后再考虑移植给 Windows，甚至根本不移植给 Windows，因为 Windows 的机器一般没有足够的能力运行这样的程序。你不要以为只有 Windows 才有 PSpice, UNIX 的 HSpice 要好得多，而且可以运行在大型主机上。当然它们不是免费的，但是它们值那个价钱。

但是 Windows 下有些东西在 Linux 下没有很相似的，或者你找到很多类似的，但是它们每一个比起 Windows 的那个程序都要差很多，那么原因有两种可能性：

有一个完全类似的程序，但是由于它乍一看不漂亮，被你忽略了。

而其它程序虽然看起来很漂亮，但是它们是一些初学编程的人写的。现在由于 Gtk, Qt 的诞生，Linux 下开发图形界面程序极其简单，很多初中生甚至小学生都可以随手编出一些漂亮不中用的程序。如果你整天寻找这样的程序挑来挑去，永远也找不到你满意的。当然也有一流的程序用 Gtk 和 Qt，比如 GVIM 就可以用 Gtk 作为图形界面，我还知道 Synopsys 一些程序用了 Qt。

我曾经也犯过这样的错误，从外表区分一切。结果优秀的 FVWM, lftp, Mutt, wget 都被我忽略过。当我找回它们的时候，我是那么的羞愧不已，它们现在都是我的朋友 :) 我第一次看到 FVWM 觉得它只不过是一个有很厚很难看边框的东西。可是现在，我的同学看到 FVWM 都说：“哇！真漂亮。”

有另一种完全不同的方式可以达到相同的目的，甚至更好

很多人很关心 Open Office, Star Office, AbiWord, ... 他们多么盼望有一天某一个 Linux 程序能够完全兼容的打开一个复杂的 doc 文档。但是你永远也不可能有那一天。为什么呢？因为微软为了占有市场，必定不会让其它系统的程序能够完全兼容它的文档格式。它一定会不断变化 doc 文档的内部结构，隐藏一些秘密，让其它公司的程序打开 doc 文档时总是有某种问题，从而你必需购买 Microsoft Office 和 Windows。

你应该想一下，那么多的高智商的大学教授，科学家，学生，他们用的都是 Linux 或者其它类型的 UNIX，他们没有 Word 可用，怎么处理文档呢？这么多年没有一个像 Open Office 的程序出现，难道大家没有办法写文档吗？

显然不是这样。你看看那些高水平的学术杂志，论文，那些大学教授的网页，那些漂亮的幻灯片，它们是什么做的？原来 UNIX 用户早就有非常方便的 troff, LaTeX, SGML 等东西可以处理文档，而且它们比起 Word 都要高明的多。Word 显然被这些大拿忽略了，以至于很久以来没有人想在 Linux 下开发一个类似 Word 的程序，除非某些公司想抢微软的饭碗。

很多人留着 Windows 在硬盘上的原因无非是为了用 Word 和 PowerPoint。我见过一个教授，他的 Windows 笔记本电脑上除了 PowerPoint 什么都没有。有一天演示的时候，他指着堆乱字符说："对不起，这是一个公式……怎么每次都是这样……" 其实有比 PowerPoint 好几百倍的东西可以制造幻灯片，你可以用最简单的方法制造世界一流效果的论文和幻灯片。你待会儿可以看看我的TeX网页，你就会知道为什么我可以完全离开 Windows。

Windows 能做的那些没用的事情 Linux 永远做不好

电脑游戏

有些人说 Linux 下不能玩 Windows 下所能得到的所有游戏。的确，Linux 下虽然也有少量的游戏，比如 Quake。但是它没有 Counter Strike[9], 没有 Star Craft, ……

并不是说电脑游戏不该玩，但是应该适可而止。电脑是用来处理事务，帮助你学习，解决问题的工具，而不是一个玩具！整天沉迷于电脑游戏中，而不出去感觉外面的世界，你会变得越来越冷酷，越来越缺乏人情味。你与真实的世界越来越远。

你可以在 CS 里杀人，你可以在 Tomb Raider 里探险，你甚至可以在 Tony Hawk's Pro Skaters 里滑板…… 但是 It's not real！你虽然有很高的"反恐技巧"，但是遇到歹徒的时候，你是那么的怯懦；你虽然控制 Laura 伸手敏捷，但是你打篮球的时候怎么总是被人断球？你虽然可以轻易的在 THPS 里作出一个 "360 kickflip to hangten grind to fakie"，但是你踩在自己的滑板上的时候还不会 ollie！

说回来，如果你偶尔玩一下电脑游戏未尝不可。但是世界上有远比 Windows + PC 更好的游戏方式。Sony 的 PlayStation2, SEGA 的 DreamCast, Nintendo 的 N64，Namco 的街机……每一个都比 Windows 游戏精彩，每一个都有如此高的3D性能，以至于 Pentium4, Itanium + GForce4 都无法与它们比美！

Linux 的用户们都是关心解决世界的关键问题的份子，他们哪里有时间用自己的机器来玩游戏啊？他们每天用Linux高效的做完自己的工作就到阳光下享受自然去了。要玩游戏也是玩一些类似推箱子，贪吃蛇之类的智力小游戏。所以，你知道为什么 Linux 几乎没有游戏了吧？:)

"整理硬盘，优化系统"

这是一个非常有意思的话题，仅次于有关“病毒”的话题。相信很多 Windows 用户都有整理硬盘的经历。在很多 Windows 用户眼里，“硬盘用久了，会出现碎片，速度会减慢，需要一个程序来整理，整理硬盘的时候不要做其它工作”，这好像是天经地义的事情。

我也曾经津津有味的看着 Norton Defrag 一点一点的把我的硬盘排序，调整，用图形的方式显示出来，然后报告：“100% 没有碎片。你的硬盘现在已经达到最佳状态。” 我现在才发觉我那时是多么的幼稚。

Linux 和 UNIX 用户似乎从来没有“整理硬盘”这种说法呢？你觉得很奇怪吗？如果你觉得很奇怪，那说明你的思想在某种程度上被微软的垃圾程序禁锢了。你需要明白，UNIX 的大型主机很多必须是一天24小时，一年365又1/4天不停运转的，要是每个星期都要整理一次硬盘，在整理的时候几乎不能干任何事情，那是绝对行不通的！

Linux 机器根本不用整理硬盘，这就是为什么没有看到过 Linux 用户整理硬盘。Linux 的文件系统是比 Windows 的 FAT, FAT32, NTFS 高明得多的文件系统，它们不但可以对文件设置权限，实施完全的保护，而且可以“越用越整齐 ”，“越用碎片越少”[10]！你应该把文件大部分放在 Linux 的分区，而不是 Windows 分区，因为它比 Windows 分区可靠得多。

还有更滑稽的事情就是有很多“Norton System Doctor”，“Windows 优化大师”，“超级兔仔注册表魔法” 之类的程序存在，而且价格昂贵。似乎一个操作系统本来应该有很多问题，需要别的厂商做程序来“优化”它，而且为了得到优化，你需要付钱！这些问题 Linux 根本就没有，所以不需要什么优化。Linux 内核本身就是高度优化的。

IDE

有些人在抱怨为什么 Linux 没有一个良好的 IDE 开发环境。Linux 现在已经有一些 IDE 了，但是总是有很多问题。你是不是正在寻找，正在期望 Linux 某一天可以有一个VC那样的开发环境？你有没有发现你正在进入微软给你设下的怪圈？你为什么一定要用 IDE？你说：“IDE 开发迅速，调试方便，适合大型程序……” 那说明微软的程序在你脑子里已经比较根深蒂固，你需要好好清醒一下了，看看我来告诉你。

高明的 UNIX 程序员不用 IDE，IDE 从来就是给初级 Windows 程序员用的。

你看看大型的 UNIX 程序，包括 Linux 内核，各种网络服务程序，Xwindow 程序在内，哪一个是 IDE 搞出来的？我们实验室的 EDA 程序也没有一个是 IDE 弄的，我还知道 Candence, Synopsys，Mentor 的高性能的图形界面 EDA 程序也都不是 IDE 写的。你信不信，微软的人在写 Windows 本身的时候也根本不用 IDE。微软内部程序员最喜欢的编辑器其实是 VIM，用 VIM 的微软程序员上次向乌干达的可怜儿童捐助了1000多美元，这是值得称赞的。

有一次某杂志采访一些出名的 Linux 内核程序员，包括 Linus 在内，没有一个人用 IDE，有的人用 VIM，有的用 Emacs，只有 Linus 说 “GNU Emacs is evil”，但是其实他用的是一种跟 Emacs 有同样键绑定功能的 MicroEmacs。大家都是用编辑器编辑了程序文件，然后用 make

这样的自动工具调用 gcc 编译器完成编译工作的。甚至高级的 Windows 程序员也不用 IDE，他们可以从命令行调用 cl，nmake 来编译自己的程序。虽然这样的 Windows 程序员很少，但是他们却是最了解 Windows，最高明的 Windows 程序员。

为什么 UNIX 程序员不用 IDE？明白了这个道理你就能体会到 UNIX 的设计思想了。首先，一个 IDE 集成了编辑器，编译器，汇编器，调试器，跟踪器…… 这个编辑器功能肯定比不上 VIM 或 Emacs，编译器比不上 GCC，汇编器比不上 as，调试器比不上 gdb, ddd, 跟踪器比不上 strace, ltrace, truss。你得到的是一套整合的低能的程序。如果你对调试器的功能不满意，你只好换用另外一套 IDE，但是这套 IDE 的热键，菜单，编辑器功能，按钮…… 跟原来那个有很大不同。你不得不花很多时间来熟悉新的环境，而不能保持原来的某些东西。

而在 UNIX 下就不一样了。你可以用你最喜欢的 VIM 编辑程序，你在 VIM 里可以调用 GNU make，make 可以调用 gcc, ld, ... make 的出错信息可以被 VIM 捕获，VIM 能帮你在源程序里定位。你如果喜欢 icc, 你可以让 make 用 icc 而不是 gcc。你如果觉得 gdb 跟踪变量时比较麻烦，你可以用 ddd 来显示各种数据结构之间的关系。你还可以在 Emacs 里调用 gdb，那样就可以同步显示源代码了。而且 VIM 和 Emacs 还可以编辑很多其它东西，比如信件，LaTeX 文档，HTML，配置文件…… 你不用另外找一个什么编辑器来干这些杂活了。很多程序比如 Mutt, tin 都可以在内部使用 VIM，这样就更方便了。实际上 make 在其它方面还能帮你很多忙，我的每一个比较大型的 LaTeX 文档都是用 make 维护的。

Linux 能干的高精尖的事情 Windows 都干不了

当然有很多事情是Linux/UNIX的专利了。因为 Windows 只能装在 PC 机上，好像以前也有 Alpha 可以使用 Windows NT，但是就是没见到有人用。PC 机的能力是很低的，像我们编程序处理 NP-Hard 问题的人，用 Windows 的机器显然速度不够，而且有时一个问题算上几天甚至几个星期，Windows 机器是以“死机”著称的，我们怎么能放心？

所以几乎所有科学计算程序，EDA 程序，高性能图像处理程序都不是 Windows 的。他们有时也会移植一些给 Windows，但是常常降低那些程序的能力。你比较过 Windows 版本的 Mathematica 和 Linux 的有什么区别吗？

IBM 制造的最大的并行计算机有 8000 多个处理器，Windows 不可能有能力管理这么多处理器，它用的是什么操作系统？答案是 Linux。

《泰坦尼克号》电影里的三维动画，那么细腻逼真，Windows机器能做出来吗？不行。那也是 Linux 机器做的。

民航总局用来训练地情人员的虚拟现实训练设备，Windows 当然无能为力。那都是商业的 IRIX 机器。

UNIX 是最早支持 TCP/IP 网络协议的系统。它上面有很多可以互相协作的网络服务程序，它们经过多年的使用和修订，已经达到比较完善的程度。而就在1997 年，微软的比尔盖茨还在扬言：“Internet 是没有前途的。” 微软的这个“远见卓识”大家应该都已见识，它后来加上的网络服务程序IIS漏洞之多，让公安部都频频发出警报，大家也是见识了的。

其实你知道了，Windows 没有一样有用的事情能比 UNIX 干的更好。

Linux 干不了的有用的事情 Windows 照样干不了

当然 Linux 不是万能的。它也有不能干的事情，电脑也有干不了的事情。但是 Linux 干不了的事情，Windows 肯定也干不了。这些事情就是我们需要探索，需要努力的事情了。在你探索的过程中，Linux 必定是你的好伙伴。

不要把Linux和Xwindow掩盖起来！

不要把我们的用户当成傻瓜。

什么？你早就知道 Windows 是垃圾？噢！你怎么不早说呢！害我废话这么多。嘿嘿。

“好了。你知道 Windows 是垃圾，你现在用什么？”

“Linux + Xwindow”

“那我问你，Xwindow 是什么样的？”

“不就是跟 Windows 差不多吗？只不过 'Start' 按钮比较方，而且上面不是一个 Windows 标志，而是一个脚丫子[11]。点击一下居然还有很漂亮的中文菜单。我喜欢！”

“你知道什么是‘根窗口’吗？”

“不知道。从来没听说过呢？”

“根窗口就是遮盖整个屏幕的那个最大的窗口。”

“哪儿有什么窗口啊！我没有看到呢？”

你发现了问题吗？这些 Linux 用户说是在用 Linux 和 Xwindow，但是他们对 Linux 和 Xwindow 几乎完全不了解。很多人用了那么久 Xwindow 都不知道根窗口是什么东西，不知道其实按钮也是窗口，不知道窗口管理器和其它程序有什么关系，大家都以为窗口上面的按钮是程序自己放上去的，不知道窗口的“class name”，“resource name”是什么东西。他们也不知道 .Xdefaults 是用来干什么的。特别是他们很多人都不知道 Xwindow 的字体是如何命名的，什么是 fontset，有了一个新的字体也不知道怎么安装。

他们被遮在 Linux 之上的一层一层的包装迷惑了，他们等待有图形界面的工具来帮助完成一切事情，他们认为 Linux 跟 Windows 一样，只是麻烦一点。他们知道 Linux 内核很好，但是他们感觉不到 Linux 和 Xwindow 在操作层面的天生的先进性，随后不久就把 Linux 完全删除掉了。你发现没有，要用户理解 UNIX 和 Xwindow 的操作层面的先进性，才是留住用户的最好办法。如果用户体会不到操作时的方便和高效，内核再好他们也不会理会。

但是用摹仿 Windows 的作法来吸引用户，永远会失败的。因为 Linux 如果摹仿 Windows 那一套低效率的方式，那么 Linux 的这套“低效率方式”永远比不上 Windows 的那一套“低效率方式”。那么用户就会说：“这个 Linux，没有一样比的上 Windows。”

Linux 天生就是继承了 UNIX 的高效的工作方式，为什么我们要把它掩盖起来？我们为什么只告诉用户 KDE 的菜单怎么用？我们为什么不能像早期的 Xwindow 书籍那样第一节就告诉用户什么是 X server, 什么是 X client，什么是 Window Manager, 什么是根窗口。第二章就告诉用户窗口有哪些属性，什么是 classname, resource name, hint，怎样使用 .Xdefaults, xrdb ……

在这里我又不得不说一下那些 Linux 的发行公司和写书的人，他们把 Linux 和 Xwindow 包装起来，却没有从基本上告诉用户 Xwindow 的工作原理。很多书籍讲授的层次就是在 Gnome, KDE 的菜单操作的层次，靠大量抓图来占篇幅，“繁荣”Linux 书籍市场。

现在很多人已经把能够利用别人的库写出一个好看的程序作为自己编程水平的象征。在这个“图形化”，“可视化” 的年代，你如果还在用 troff, LaTeX 写文档，你还在用 VIM 自己编辑 HTML，用 Mutt 处理邮件，你还在用文本模式的 gdb 调试程序，你还在用 Xlib 写程序, 你还在用 tin 上 USENET，你还在自己写 Makefile，写机器代码，你还在玩 Clossal Cave 这样的字符模式冒险游戏，那你就是老古董。

其实这种思想是错误的。虽然你是一个坚决的 Linux 支持者，但是你的思想是 Windows 的思想。你认为图形界面，菜单，按钮就可以解决一切问题，就可以给你高效方便。你还是没能摆脱微软给你的潜移默化的东西。你其实离不开 Windows 那样的环境，你迟早会删掉自己的 Linux。

> [9] 已可以通过 Wine 良好运行
>
> [10] 也不尽然……总体来说，Linux 的文件系统要健壮的多
>
> [11] Gnome 徽标，多数发行版都会替换为自己的，通常是你不会有机会看到这个“脚丫子”的

GUI vs. CLI

做一个坚定不移的“两面派”

大家看到这个标题是不是热血沸腾？两派大虾都可以围攻我了：

GUI派用户：“哇！我一看你这小子就是 CLI 的。要不然自己写什么 Makefile？用什么 Mutt？”

CLI派用户：“切～ 你还用 X！高手都不用 X。你是 GUI 那边的。”

可怜的我：“555～～ 你们都不要我～～ GUI 和 CLI 就那么水火不容吗？”

计算机界这样的门派之分还很多。很有特点的就是 CLI 和 GUI 了。CLI (Command LIne[12]) 的狂热份子声称永远不用 X。我上次在实验室看到一个同学用一个 SecureCRT[13] 登录到 Sun 机器，然后用一个 vanilla vi 编辑程序，我建议他启动一个 GVIM 过来显示在 Exceed 上

可以有语法加亮。但是他坚决反对，说："高手不用X。你想想，要是我在一个很慢的网络连接怎么用 X？而且好多服务器没有装 X 程序。"

但是我们实验室的网速可够快，Windows 机器都有 Exceed 啊，而且 Sun 机器有全套 X 客户程序包括 GVIM。他说他是 CLI 的坚决拥护者，但是他却在用 Windows，他后来打开了好几个 SecureCRT，每次从文本框输入地址，用户名和密码，从下拉菜单选择 "SSH2"，然后点击"Connnect"。他还不断的夸SecureCRT是"网络管理员投票选出的最受欢迎的登录方式"。老天，SecureCRT 本身就是个 GUI 啊，他其实没有明白 Xwindow 的好处。

你说我是 GUI 的？我虽然很少在 console 下工作。但是我对 bash, VIM 很熟悉，我可以让 bash 按照我的键绑定方式来工作。我可以在 rxvt 里使用 Mutt 来收发 email。我的每个桌面上都常常堆放着一打不同大小的 rxvt。我用 VIM 编辑 LaTeX。我自己写 Makefile 来维护 LaTeX 文档。我有时用 mpg321 来放 mp3。我上BBS用的我自己写的 expect 脚本。 好了，CLI 派的朋友可以收我做盟友了 :)

你说我是 CLI 的老古董？我的 FVWM 被我配置为可以"手写操作"，我只要画一个"r"就可以启动 rxvt，我只要画一个 "U" 就可以启动 GVIM，…… 我用 GVIM 语法加亮模式编辑程序，我用 Mozilla 浏览网页，…… GUI 派的现在好像认我做朋友了 :)

好了。CLI 派的朋友，虽然我很喜欢命令行，但是我有时在屏幕上左右画一下就可以执行:

```
Module FvwmConsole -terminal rxvt -geometry 45x5-0+0 \
-bg gold -fg midnightblue  \
-fn "-adobe-courier-medium-r-*-*-14-*-*-*-*-*-*-*
```

你是不是现在又想把我逐出师门？

GUI 派的朋友，虽然我很喜欢窗口。但是我可以在 FvwmConsole 里输入：

```
All (rxvt) MoveToDesk
```

把我所有的 rxvt 移动到我现在工作的桌面。"这家伙，怎么这么快就叛变了！"

其实何必分什么 GUI 和 CLI，UNIX 和 Xwindow 都是工业标准，它们从设计那天开始就有非常灵活的用法，各个程序，不管是 GUI 还是命令行的都可以互相协作。UNIX 和 X 是一家，何必搞的那么偏激，非此即彼？你从我上面的行为可以看出 GUI 和 CLI 的模糊界线吗？我就是坚定不移的"两面派"。

> [12] Command Line Interface
>
> [13] Putty 更好用

UNIX 是简单的

“我相信简单就是最好，如果太复杂，我是不能理解的。” —Seymour Cray

很多第一次用 Linux 的人会惊奇的发现，Linux 的程序居然不用“安装”就可以运行，程序拷贝到随便那个目录都可以用，而不是一定要占用你第一个分区的空间。程序的设置只是一些简简单单的文本文件。你根本不需要什么“注册表修改器 ” 就可以改变系统的设置。这就叫做简单，但是简单就是美。虽然这只是 UNIX 简单性的一个肤浅的认识，你已经体会到了某些东西。

但是简单并不意味着功能弱，并不意味着落后。相反，简单意味着强大，意味着生命力。

我不会再继续阐述我理解到的“UNIX 的简单”，因为这个需要自己去体会。

UNIX 是永恒的

有人说：“Plan9 会取代 UNIX，Mach 会取代 Linux 内核。”

但是你如果是一个深入体会了 UNIX 的人，你就会知道：UNIX 的思想是永恒的，不管时过境迁，Plan9 是否代替 UNIX，UNIX 的灵魂都会在 Plan9 身上现形！

我为同一个设备写过 Linux 内核和 Windows VxD 驱动程序。写 Linux 驱动程序时，我对 UNIX 设计的完美的一致性，远见性所折服。UNIX 用同样界面的 read(), write() 系统调用就可以对不同的对象：普通文件，设备文件，管道，管道文件，socket，…… 进行统一的读写操作。我跟本不需要写一个测试用的应用程序就可以对我的设备驱动进行测试，因为 cat, cp, dd, 它们也使用了同样的 read(), write()，设备和普通文件在应用程序眼里没有区别。在那个还没有 Smalltalk, 没有 C++ 的年代，UNIX 的设计者已经使用了所谓的 “面向对象方法”。对，C 语言也可以实现面向对象。

UNIX 的系统调用几十年都没有很大变化，这非但不是顽固，不进步的象征，反而是 UNIX 的远见卓识的体现！这就跟 TeX 程序几十年都不变的情况差不多。这些才是真正的永恒的 master piece! 你应该改变所有软件都必需从 0.1, 1.0, 1.1, 1.2, 2.0, ..., 3.0, 3.1, 95, 98, 2000, XP, ... 不断升级的想法。

Windows 就不同了，它在最开头只是一个 DOS 之上的图形包装而已。后来为了兼容以前的糟糕设计，不得不加上很多累赘。我写 VxD 驱动程序的时候就深有体会，Windows 95 程序对设备的操作只有用 DeviceIoControl，我不得不写了两个应用程序来对设备驱动进行测试。Windows 内核的不一致性和隐密性使我非常恼火。不过 Windows WDM 驱动程序现在也有了 ReadFile, WriteFile，…… 那说明什么？那说明 Windows 在向 UNIX 学习，或者有可能是某个 UNIX 设计人员在微软打了几天临工，顺手加了几个 UNIX 的东西进去。这样做是没有用的，Windows 从一开始就是非常糟糕的设计，它的历史的包袱太沉重了，缝缝补补有什么用？它只能永远的被 UNIX 甩在身后！

UNIX 是强大的

让聪明人干任何他们想干的事情。

UNIX 的一个特点就是非常高的灵活性，Xwindow 也具有这种灵活性。这种灵活性体现在哪里呢？

UNIX 的程序一般都有很多参数，不管你现在用的着用不着，总有人需要某些参数。它们的行为很多都可以用配置文件来改变。比如 GNU bash, 通常缺省的命令行输入方式是 Emacs 方式，但是只要我编辑一个 .inputrc 文件，就可以把它变成 vi 的输入方式，而且我还可以自己绑定键序列到某些操作。我可以用 shopt 来设置它的很多特点，比如是否进行通配符扩展，是否可以把一个变量当作一个目录来cd，是否可以自动纠正某些明显的目录名打字错误 ……

UNIX 程序设计的思想是提供给用户“机制”，而不限制用户制定“政策”。这是一个重要的尊重用户的作法。

我们再来看看 Xwindow。Xwindow 是一个出色的设计，它把显示服务器和客户程序分开。一个显示上既可以显示本机上的程序，也可以显示别的机器上的 X 程序，而它们都遵守你的窗口管理器的统一指挥，它们之间可以方便的传送剪贴版数据，各种事件 …… 比如有时我的 XFree86 上会出现四个不同机器上的 XTerm，两个不同机器上的 GVIM，…… 它们统一受本机上的 FVWM 指挥。

Xwindow 程序都具有很多很多命令行参数和 resource 参数。你可以随意的在命令行或者 .Xdefaults 文件设置所有的颜色，字体，尺寸…… 而且如果你用 xrdb 把 .Xdefaults 导入到根窗口，那么其它机器上没有经过配置的同样的程序，显示到你的机器上的时候也会遵守同样的外观规定。

Xwindow 的窗口具有 Property, 也就是一些可以自己定义的共享数据(原子)。正是因为这些 Property 的存在，使得 Xwindow 具有无比强大的生命力。X 的窗口管理器和其它客户程序之间并没有统一的协议，但是后来出现了 ICCCM(客户程序间通信规范)，这个规范就是通过 property 定义的。现在又有人定义了一套“扩展的窗口协议(EWM Hints)”，使得 Xwindow 可以具有某些 Windows 的特征，比如一个工具条程序可以告诉窗口管理器：“这个屏幕下面被我占据了24个像素的空间，你最大化程序的时候不要越过这个界线。”

一个强大的窗口管理程序比如 FVWM，它收到这样的提示时，可以答应工具条程序的这个要求，也可以不答应。一切选择的权力在于谁？当然是用户了！一切窗口乖乖听话，FVWM 给予用户最大的尊重。

你想想，是不是有些 Windows 程序常常弹出一个窗口要你选择 "Yes or No"？你不点击它它就不下去。你觉不觉得你的程序在侵犯你的尊严？你是一个人，一个智慧的生物，怎能受到一个程序如此的待遇？

还有就是很多 Windows 程序把人当成傻瓜，而它是“智能程序”。比如，有一个程序就是喜欢把你的每句话第一个字母都变成大写，我不说它是谁了，你遇到的时候就知道了。如果连“一句话开头一个字母要大写”这么明显的问题都需要程序帮你纠正的话，人脑还用来干什么？况

且如果你故意想要不大写的话，那就更麻烦了，我楞是没有从它那一大堆菜单里找到怎么关闭这个愚蠢的选项。

只有符号才能完全操纵计算机

我们来说说很多初学 Linux 的用户。虽然他们在用 Linux，但是他们打心眼儿里是觉得 Windows 的工作方式好，他们希望 Linux 有一天能“像Windows那样”。你说：“我鼠标一点，我菜单一拉，…… 就可以完成我的操作。” 但是我要告诉你：“Linux 从来没有摹仿 Windows，将来也不会。Linux 从诞生之日起，它的工作方式就比 Windows 的先进。Linux 属于能勇敢面对符号的人。只有符号才能完全操纵计算机。”

看看优秀的 UNIX 程序，XFree86, FVWM, VIM, Emacs, proftpd, Mutt, wget, tin, ... 没有一个不是用配置文件来设置选项的。为什么这些程序没有方便的菜单可以用来配置？难道它们的设计者就那么低能，连个图形配置界面也写不出来？

当然不是。因为图形界面配置方式的能力是极其有限的，而配置文件和程序语言的表达能力却是无限的。用图形界面配置这些程序的话，如果你想达到配置文件的效果，你需要成百上千的菜单，checkbox, radio button, ... 到时候你根本没办法找到你需要修改的地方了！而各个程序的配置文件的语法都有很多相似之处，一般就是一些命令，设置一些变量，参数，…… 一旦用会了一个，其它的也就容易理解了。如果你用惯了 awk, sed, Perl，你会觉得那才是眞正的自动化啊。

鼠标虽然是很好的工具，但是它的表达能力是有限的。你不可能光用鼠标就让电脑完全明白你的意思，它毕竟只有3个按钮。看看我的MetaPost页你就能体会到鼠标的这一弱点。所以我们虽然很喜欢鼠标，但是却不能完全依赖它。

各个小程序的完美配合

这就是UNIX最重要的特点了，它就是UNIX设计的思想。让每个程序只具有一项专门的能力，然后让它们合作。Xwindow也继承了这种好传统。

这恐怕就是Windows和其它操作系统望尘末及的地方了。UNIX 程序设计之统一，配合之完美，眞使我难以置信！shell, grep, find, awk, sed, make, Perl, Emacs, vi, tin, Mutt, ... 它们是那么的具有一致性！你一旦学会了 sed 的正则表达式，其它程序基本上都能用了。你一旦学会了 vi 和 VIM, 你会发现它的操作是那么的有规律性，似乎vi的设计者在几十年前就已经设计好了 VIM 在今天的完美而统一的操作方式！而且vi的操作还体现在 Mutt, tin 等很多程序中。你甚至可以把 bash 设置为 vi 的输入方式来输入命令行，我就是这么做的。一个程序可以调用另外一个程序来得到数据，可以把数据交给它处理后返回来，可以在自己的窗口里“嵌入”另外一个程序。

在 Windows 和其它非 UNIX 操作系统中，这种合作是非常困难的。我曾经在 Windows 下使用 Perl来进行一些自动工作。但是 Windows 的文件操作，管道是如此的不稳定，程序之间基本不能合作。你别想在 Visual Studio 窗口里面嵌入 UltraEdit 编辑器，你别想用一个 expect

脚本来控制 telnet 到水木清华 BBS。

Windows 的程序都是大而全，大而杂，所有的电子邮件程序都需要自己提供编辑器，自己发送和收取邮件，自己显示邮件的附件。每一个BBS程序都提供自己的Virtual Terminal, 自己的通讯代码。每一个 IDE 都自己提供编辑器，编译器，汇编器，调试器。人们为了使用一种新的程序，需要适应所有这些它提供的界面，而不能使用自己喜欢的编辑器的键绑定，菜单组织…… 不能 DIY！

你要知道，最高级的电脑是定做的，自己想要什么什么CPU，什么主板，多少内存，什么硬盘，键盘，鼠标，显示器都是自己选择的。最高级的滑板，自己想要什么牌子的版面，什么牌子的沙，什么桥，什么轮子，什么轴承，也都是自己选的。最高级的乒乓球拍，木板，胶皮，海绵，胶水都是可以自己选择…… 而用 Windows 程序，你得到的是大杂烩，就像你去买“品牌机”，只有那么几种配置，而且附带很多你不需要的软件和服务；就像你去买组装好的滑板，你想要大一点的轮子和窄一点的板子，但是你没有这种选择余地！Windows 程序就相当于最廉价，最次的滑板。但是它却会花你更多的钱，因为一旦一个部件坏了，或者你不喜欢了，你不能另外找一个好的换掉它，你必需重新买全套配件！

而 UNIX 和 Xwindow 就是高档的“组装货”。比如我用 Mutt 的时候，我可以用 VIM 也可以用 pico 来编辑邮件，我可以用 ImageMagick 也可以用 xv 来显示附件里的图片，我可以用 lynx 把 HTML 附件转成文本嵌入窗口中，我也可以把 HTML 附件交给 Mozilla 图形显示。我可以让 GnuPG 帮我把邮件进行数字签名和加密，我也可以用其它 PGP 程序。我想让 Postfix 而不是 sendmail 帮我发出邮件，我想让 fetchmail 帮我收邮件，转发给 postfix，然后被我自己写的Perl过滤器处理…… 这一切我都可以办到！我可以选择我最喜欢的专门的程序来完成专门的工作，然后把它们结合在一起，我也可以分别得到它们的好处。

学 UNIX 绝对不是浪费时间

有人告诉我：“你看我用 Windows 什么都不用学。而用 Linux，光是安装就花了我一个星期！”

首先，我要告诉你的是，你装 Linux 花了一个星期，不是因为 Linux 不好装，而是因为你已经习惯了 Windows，对 Linux 最初难以理解而已。你想一想你最初安装 Windows 的时候呢？你花了多少时间搞明白什么是硬盘分区？什么是盘符？什么是目录？你认为 Windows 就是那么容易可以学会的吗？虽然你觉得没花时间学，但是你以前在用别人的机器的时候已经耳濡目染，自然就了解了。而且由于你想要 Linux 和 Windows 并存于硬盘上，又增加了安装难度。而且你肯定没有得到有经验的 Linux 用户的帮助，否则他们会在 20 分种之内帮你搞定。一个星期也太夸张了 :P

如果一开始用的就是Linux就没有这个问题。你想想如果你没有用过 windows，你肯定会很习惯 /etc, /usr, /usr/local ,... 而不是 C:, D:, E:, ... 是不是？如果你只用过 Linux，你第一次用 windows 时恐怕也会问：“/bin 目录哪里去了啊？”

最重要的是，你用惯了的UNIX工具，它们可以伴随你一生，而不会那么容易变化或消失。你可以永远不用再换另外的工具了。除非那个工具比你这个好的太多，而且可以完全模拟你现在的工具。

我们实验室一个60多岁的老师，用vi, cc, make, ...都几十年了，他以前的经验绝对没有白费，而且教会了我们一批又一批的学生。vi 伴随着 UNIX 的最初发行而诞生，直到今天还是世界上头两号编辑器之一！有些人的 FVWM 配置文件已经用了 10 多年，现在完全不经修改还可以用。

看看 Windows 的工具，你从 Borland C++ 换到 VC, 就必需适应新的环境：菜单不同了，颜色不同了，按钮不同了，帮助信息不同了，热键不同了，编译器参数，调试器功能也不同了，…… 那个时候恐怕花要花你很多时间去适应。当你刚刚适应了 VC, 你又要换成 VJ, PowerBuilder, C++Builder, ...

很多windows程序员都是这样，开头在dos下用 Turbo C, 然后是 Borland C, VC, C++ Builder, ...不断追赶微软的潮流。而且微软的 SDK, MFC, .NET …… 什么都在不断变化，不断出问题，又不断的在修改…… Windows 程序员不得不买又厚又重的 Microsoft Press 的书籍，看了才一个月，又过时了。今天你才学会了写 VxD，明天你就必须用 WDM 了。你不得不注册 MSDN 才能赶上 Microsoft 的步伐。很多人说：“计算机是贵族的专业。” 这就是微软一手造成的。

这些东西才是没完没了的浪费大家的时间和金钱的。这是浪费生命！我们为什么不使用从诞生就那么一致和完美的 UNIX？你需要理解先进工具的设计理念。UNIX 的工具就像我们用的汽车，它的离合器，油门，刹车，方向盘，后视镜，永远都在同样的位置。用惯了的话，你对你的汽车的每一个部件都会了如指掌，甚至你自己都可以修车了。这难道不好吗？

有人说：“你说我们需要了解 UNIX，难道你要开车还必须了解汽车的结构吗？” 你去问问开车的司机，哪一个不了解汽车的结构的，那他的驾照就是混来的。你难道想要傻瓜型的“微软牌汽车”吗？我们来看看：

你买的微软牌汽车最开头只有一个座位，每加一个座位你得向汽车公司付钱。车上的防撞气囊不时会冒出来，说是为了你的安全。每开100英里要大修一次，每过一年要换一次引擎。附带的，你还必须换用由微软汽车公司指定的石油公司提供的新型号的机油。你的车出了问题，但是法律规定，你不准私自折开你的汽车来修理，你必需到微软汽车公司指定的维修点去，需要付相当多的钱才能修好一个小毛病。

最可气的是，你每换一个型号的微软牌汽车，它的刹车和离合器都在不同的位置，你需要重新去考驾驶执照。如果这辆汽车在途中刹车失灵，你受了重伤，你也不能状告微软汽车公司，因为你买来汽车之后必须签一个合同，说 “由于微软牌汽车对你和家人造成的一切死伤，微软概不负责。”

怎样完全用 GNU/Linux 工作

说了这么多 Windows 的不好。我还没有告诉你我怎么用 Linux 处理有些必要的事情。

半年以前我由于中文老是配置不好，一直是双系统，不时需要重起到 Win2k 来处理汉字。后来我找到了 miniChinput, XSIM 和 SCIM 输入法。这下可以处理汉字了。而且 VIM 和 Emacs 对汉字支持越来越好。我的大部分文本是用 VIM 编辑的，包括程序，信件，网页，LaTeX 论文，MetaPost 绘图语言。

我不用 Word 这样的程序写论文，而是用 LaTeX，因为这是世界上效果最好，最方便的论文工具，是大多数学术杂志要求的格式。幻灯都是用 ConTeXt 做的，用起来很简单，而且效果非常漂亮。你可以看看我的TeX介绍。

至于绘图，你可以用很多可视化的工具，比如 xfig，dia。但是对于我来说，任何可视化的工具都不能完成某些任务，我需要一种可以精确描述图形的语言。我找到了MetaPost。它简单又好用，而且效果是世界一流的。我的插图，如果不是图像，都是 MetaPost 画出来的。

我曾经抱怨 mozilla-mail 经常突然消失，损坏我好几封快要完成的信件。后来我发现 mozilla 的邮件处理程序确实是不稳定的，功能又弱，有经验的 UNIX 用户都不用这样的程序。Mutt 是一个非常稳定可靠的 UNIX 邮件处理程序，而且功能非常强大。

我曾经为 Gnome 和 KDE 的不稳定而烦恼。现在我找到了非常强大的 FVWM。KDE，Gnome 也能和 FVWM 一起工作。虽然 Gnome 和 KDE 总体不稳定，但是某些部件程序还不错，很多 gtk, Qt 的程序也很不错，它们很多都是可以独立于这些桌面环境运行的。

Linux 有很多强大方便的工作方式是 Windows 没有的，或者有类似的东西，但是很差劲或者用起来不方便。比如 ssh 服务，rsync，cvs，expect ……

结论

我写这么多的目的是什么？我希望喜欢 Linux 的朋友，完全清除微软和 Windows 灌输在你脑子里的谬论，别再相信它们所谓的“新技术”，别再追赶 Windows，因为追赶 Windows ＝ 倒退。马克思有一个思想很重要，“新生事物并不一定是在最近出现的。” UNIX，Xwindow, TeX 虽然都比 Windows 先出现，但是它们才是先进生产力的代表。我们要清楚的认识到什么才是真正的现代化，什么才是真正的自动化。

消除学计算机很难的幻觉，勇敢的拿起像 bash, FVWM, TeX, VIM, Emacs, Mutt …… 这样强大的程序，勇敢的面对符号。不要再埋怨“Linux 为什么不能像 Windows 那样”，不要再浪费时间试用这样那样的程序，不要再忙着升级。你需要理解 UNIX 的工作方式，因为那是大多数科学家的工作方式。 Linux 可以成为你的好朋友，你需要认识它，了解它，信任它，才能完全的靠它来高效的工作。当然，在游戏机，手机，掌上电脑里，或者在用电脑来娱乐的时候，用一些“傻瓜软件”还是不错的 :)

我希望小学，中学的计算机老师能够提高自己的素质，在孩子们的启蒙教育中充分利用 Linux 神秘的特点，引起孩子们对数学，对符号的好奇心。诱导他们用计算机来解决世界上的有趣问题，而不要把教学的范围局限于计算机的操作和它自身的问题。

附录: 我用来处理日常事务的 Linux 程序

好了好了。我知道你发现自己应该转向 Linux，你很后悔当初为什么中了微软的邪。但是不要着急。因为这些东西本来只是工具，它们是用来完成你的主要任务的辅助而已。你以前选错了工具，这不要紧。你还是拥有你自己原来的专业技能，那才是最重要的。工具的东西只有慢慢适应转换，不能一蹴而就，否则你会感到非常没意思，甚至放弃。

如果你只想做一个像我这样的普通用户，主要目的是用 Linux 来完成自己的任务，那就可以不用系统管理员或者网络管理员的标准来要求自己，因为当一个系统和网络管理员确实很辛苦。这里我对实验室的网管同学鞠一躬，谢谢你的指点和帮助 :) 不用把你的机器当成网络服务器，不用开放没有必要的服务，设置好 ssh, ftp 已经足够了。这样会省去了解很多没必要了解的东西的时间。不用过度考虑“安全”，因为 Linux 缺省已经很安全了。不过你有兴趣了解更多那也无妨。

下面给出一些推荐使用的可以处理一般事情的程序。至于你的专业上要用到的科学和工程软件比如 Matlab, Mathematica, Maple, HSpice, Design Compiler, …… 还有其它物理上的，化学上的，生物上的 …… 都必然有 Linux 和 UNIX 的版本。当然他们很多不是免费的，不要总是觉得什么都应该免费，自由不等于免费。它们是经过很多人辛勤劳动的产物，是可靠的程序，它们物有所值。

下面列出我常用的一些 Linux 程序。一个列表里可能有很多，那是为了方便你来选择，我列出了比较信得过的。但其实很多只有第一个是我眞正在用的，我不喜欢试用程序。我不是一个合格的网络管理员，我的服务器都只设置了我自己需要的功能，那样可以省去我很多麻烦 :P

Shell: bash。它结合了 csh 和 ksh 的优点，并且有 readline 功能，你可以随意绑定自己的键盘。

编辑器： VIM, Emacs。

程序开发： GCC, make, ld, Scheme48, j2sdk, Perl, Python, Tcl/Tk ...

论文，幻灯工具：LaTeX, ConTeXt

绘图工具：MetaPost。这个语言太强了，以至于我只用它了。你不熟悉的话可以用 xfig, dia 来画一些流程图之类的图片。

图像处理：ImageMagick。其中的 import 程序可以屏幕抓图，convert 程序可以转换图像格式，display 可以显示图片和简单编辑(缩放，换质量，转格式，简单绘图，简单虑镜)。通常我就这么点需要。如果你要更强大的图像工具可以用 Gimp, 它几乎和 Photoshop 差不多。

自动管理工具：make。我可以用make来自动编译程序，自动编译文档，自动更新插图……全自动，而且不会重复劳动。

数值计算程序：SciLab。这个程序基本上可以代替 Matlab。

代数计算程序：MAXIMA。这个程序基于世界上最老的计算机代数系统之一: 由美国能源部(DOE)发行的 MIT Macsyma 系统。它是用 Common Lisp 实现的。很多现在的符号计算程序比如 Maple 都从 MAXIMA 身上学到很多东西。它现在经过 DOE 批准以 GPL发行，永远是一个自由软件。

加密程序：GnuPG。我的 PGP 密钥就是它搞出来的。

打包，压缩程序。什么都有: tar, gzip, bzip2, zip, rar, ...

虚拟光驱程序。Linux 不需要虚拟光驱程序，直接 mount 就行了。

ftp 服务器：proftpd, vsftpd。proftpd 功能很强，但是我只用了最简单的一种设置。

WWW 服务器：apache。(我一般没有开)

ftp 客户程序：lftp，ncftp。它们都是文本方式操作的，但是比起图形界面的方便的多。比如 lftp 几乎具有 bash 的所有方便功能，Tab 补全，bookmark, queue, 后台下载，镜像...... Linux 也有图形界面的 ftp 客户程序，但是大多不稳定，有很多问题。这就是很多人抱怨 Linux 不如 Windows 的一个小原因。还有很多人用 Wine 模拟 Windows 的 leapftp，其实 lftp 比 leapftp 好很多，你需要的只是适应一下。

自动下载工具：wget。它非常稳定，有一次我下载一个程序，用 IE 和 Mozilla 下载回来的文件都是坏的，最后还是 wget 可靠的传输了数据。用它甚至可以镜像整个网站，比起 WebZip 这样的 Windows 程序强多了，而且不会因为你不付钱就在下载回来的网页里强制插入广告。

虚拟终端：rxvt, xterm, gnome-terminal, mlterm, ...

X server: XFree86 [14]

窗口管理器：FVWM。编译加入了 libstroke。

中文输入：XSIM。被我修改过以适应 FVWM 的需要。另外推荐你还可以用 SCIM。

email 处理：Mutt + Postfix + fetchmail

看 PDF, PS, DJVU 文件：Acrobat Reader, xpdf, GhostScript, gv, djvu工具包和 netscape 插件。

看CAJ文档。我从来不看CAJ之类的文档，如果找不到PDF或PS，直接去图书馆借最好。

看网页：Mozilla, Phoenix, lynx。Mozilla-Xft 的显示效果比 IE 好很多。

英汉字典：IBM智能词典，星际译王。

编辑网页：我用 VIM 直接写 HTML。你如果想要图形方式的可以用其它的比如 screem, BlueFish。

登录其它 UNIX, Linux 机器：openSSH, telnet。 我喜欢用 openSSH 把其它机器的 X 程序通过 ssh 加密的隧道传到我机器上显示。

登录 Windows2000 server 的 display service: rdesktop，...我有一天试了一下，不错。后来就没有用过了。

同步程序：rsync。我用 rsync 通过 ssh 来跟某些机器同步数据，或者做自己机器上不同目录间的同步。

上BBS：rxvt(或任何一种终端) + telnet + chatbot(helloooo 机器人的程序)

QQ, ICQ: 我没有 QQ[15] 或 ICQ。不过你可以用 Gaim, 它同时支持 QQ, ICQ 和很多其它的即时通信方式。ICQ 用户也可以用 Licq。

放录像：MPlayer, RealPlayer。MPlayer 太好了，直接就可以放 VCD, DVD, divx, wma, wmv ... 用 Windows 的同学都很羡慕我，说 Windows 要放这个需要大堆插件。 rm 最好还是用 realplayer 放，它也是免费的。

放音乐： xmms(mp3,ogg都可以), mpg321(放mp3), ogg123(放ogg)。mpg321 不如 xmms 管理音乐文件那么方便，但是有时我还是用 mpg321 放 mp3 作为背景音乐，因为懒得开一个 xmms窗口 :)

游戏：我觉得 KDE 的那个 ksokoban(推箱子)，很好玩 :)

看 Word 文档。请 Word 用户把文档全部转为 PDF 或 PS 再给我，文档里没有特殊的格式干脆就用文本吧，何必那么麻烦。以前很奇怪的是，通知里本来没有什么特殊的格式居然还要发doc附件的email。现在好了，我们系发通知都用文本，PDF，甚至图片了 :P

其它程序：还有很多我需要用而你不一定用得着的。比如，Doctor Scheme, Scheme48, Scsh, kawa...这些程序只有 Doctor Scheme 有 Windows版本。还有很多幕后工作但是你一般不察觉的：xinetd, telnetd, sshd, crond, atd, lpd, ... 他们都比 Windows 的对应者强的多，或者根本没有对应者。

---

[14] 现在通常都是 X.Org

[15] 已经有了 QQ for Linux

第 7 章 病毒

目录

引言

这是一个非常流行的言论：

Linux 下的病毒少，是因为 Linux 的使用者少，骇客显然不愿意浪费气力去攻击没有人使用的操作系统。

您可能已经知道了，互联网上用作重要用途的服务器，其中很大一部分是 Linux 系统，另外的一部分是 Unix 系统：） 如果骇客能够搞掉 Linux 系统的话，那么整个互联网就会陷于瘫痪！效果似乎更好一些。

当然了，您一定会想：骇客也是人，他们也喜欢上网，兔子还不吃窝边草呢……兔子那么笨，连乌龟都跑不过……骇客们可比兔子要聪明的多了！它们总没有理由让自己无家可归吧?

是的，我承认这一点……不过他们也不一定非得把互联网干掉。很多骇客作梦都想入侵美国军方的服务器，美军服务器中的绝密数据，我想拉登大叔一定愿意以重金买下的!

如果骇客可以入侵任何一台主机[16]，他们为什么不去入侵美军的服务器，而要入侵您的电脑呢？

有些人又会认为：这样的解释不能说明什么，Linux 比 Windows 安全也许只是偶然的，或许下一个版本的 Windows 会超过 Linux。

个人倾向认为这种情况不可能出现，后面的论述基本摘自《UNIX 编程艺术》[17]

顺带提一下，与上面的观点相呼应，另一种论调也使人侧目：Windows 服务器占到了服务器操作系统xx%的份额。

或许这个现象可以用80：20法则来解释：

占服务器总数80%的 Windows 提供了服务总量的20%

请您务必注意，这只是举一个例子，Windows 服务器可能永远也不会占到服务器总数的80%！它提供的服务，以我个人的角度，我不认为可以达到20%，而且永远不会有那一天。

> [16] 我看过一篇报道，美国海军的一个港口的指挥塔，以前用 Windows，每天不得不重启主机N次，后来实在扛不住，转用了 Linux。指挥塔只不过是指挥几艘小船靠个岸，负载不大，也不是特别重要，尚且不可以用 Windows，军方的中央服务器就更不可能了
>
> [17] 这本书算是开源信仰的《福音书》，建议弄一本看看。由于这本书实在太长了，所以我就不全文摘抄了

Windows 的缺陷

尽管支持抢先式多任务处理，但进程生成却很昂贵——虽然比不上 VMS，但是（平均生成一个进程需要0.1秒左右）要比现在的 Unix 高出一个数量级。脚本功能薄弱，操作系统广泛使用二进制文件格式。除了此前我们总结过的，还有这些后果：

大多数程序都不能用脚本调用。程序间依赖复杂脆弱的远程过程调用（RPC）来通信，这是滋生 bug 的温床。

………………

Unix 的系统配置和用户配置数据分散存放在众多的 dotfiles(名字以“.”开头的文件)和系统数据文件中，而 NT 则集中存放在注册表中。以下后果贯穿于设计中：

- 注册表使得整个系统完全不具备正交性。应用程序的单点故障就会损毁注册表，经常使得整个操作系统无法使用、必须重装。
- 注册表蠕变(registry creep) 现象：随着注册表的膨胀，越来越大的存取开销拖慢了所有程序的运行。

互联网上的 NT 系统因易受各种蠕虫、病毒、损毁程序以及破解（crack）的攻击而臭名昭著。原因很多，但有一些是根本性的，最根本的就是：NT 的内部边界漏洞太多。

NT 有访问控制列表，可用于实现用户权限组管理，但许多遗留代码对此视而不见，而操作系统为了不破坏向后兼容性又允许这种现象的存在。在各个 GUI 客户端之间的消息通讯机制也没有安全控制，如果加上的话，也会破坏向后兼容性。

虽然 NT 将要使用 MMU，出于性能方面的考虑，NT 3.5后的版本将系统 GUI 和优先内核一起塞进了同一个地址空间。为了获得和 Unix 相近的速度，最新版本的 NT 甚至将 Web 服务器也塞进了内核空间。

由于这些内部边界漏洞产生的协合效应，要在 NT 上达到真正的安全实际上是不可能的。如果入侵者随便作为什么用户把一段代码运行起来（例如，通过 Outlook email 宏功能），这段代码就可以通过窗口系统向其它任何运行的应用程序发送虚假信息。只要利用缓存溢出或 GUI 及 Web 服务器的缺口就可以控制整个系统。

Unix 的健壮

Unix 至少设立了三层内部边界来防范恶意用户或有缺陷的程序。一层是内存管理：Unix 用硬件自身的内存管理单元（MMU）来保证各自的进程不会侵入到其它进程的内存地址空间。第二层是为多用户设置的真正权限组——普通用户（非 root用户）的进程未经允许，就不能更改或者读取其他用户的文件。第三层是把涉及关键安全性的功能限制在尽可能小的可信代码块上。在 Unix 中，即使是 shell（系统命令解释器）也不是什么特权程序。

操作系统内部边界的稳定不仅是一个设计的抽象问题，它对系统安全性有着重要的实际影响。

彻头彻尾的反 Unix 系统，就是抛弃或回避内存管理，这样失控的进程就可以任意摧毁、搅乱或破坏掉其它正在运行的程序；弱化甚至不设置权限组，这样用户就可以轻而易举地修改他人的文件和系统的关键数据（例如，掌控了 Word 程序的宏病毒可以格式化硬盘）；依赖大量的代码，如整个 shell 和 GUI，这样任何代码的 bug 或对代码的成功攻击都可以威胁到整个系统。

历史

大部分1980年前的 Unix 竞争者都被拴到单个硬件平台上，随着这个硬件的消亡而消亡。为什么 VMS 可以坚持这么久？值得我们作为案例研究一个原因是：VMS 成功地从最初的 VAX 硬件移植到了 Alpha 处理器（2003年正从 Alpha 移植到 Itanium 上）。MacOS 也在1980年代后期成功完成了从摩托罗拉68000到 PowerPC 芯片的迁跃。微软的 Windows 处在计算机商品化将通用计算机市场扁平化到单一 PC 文化的时期，真是生逢其时。

自1980年起，对于那些要么被 Unix 压倒要么已经先 Unix 而去的其它系统，不断重现的另一个特有弱点是：不具备良好的网络支持能力。

在一个网络无处不在的世界，即使为单个用户设计的系统也需要多用户能力（多种权限组）——因为如果不具备这一点，任何可能欺骗用户运行恶意代码的网络事务都将颠覆整个系统（Windows 宏病毒只是冰山一角）。如果不具备强大的多任务处理能力，操作系统同时处理网络传输和运行用户程序的能力将被削弱。操作系统还需要高效的 IPC，这样网络程序彼此能够通信，并且能够与用户的前台应用程序通信。

Windows 在这些领域具有严重缺陷却逃脱了惩罚，这仅仅因为它们在连网变得真正重要以前就形成了垄断地位，并拥有一群已经对机器经常崩溃和无数安全漏洞习以为常的用户。微软的这种地位并不稳定，Linux 阵营正是利用这一点成功地（于2003年）在服务器操作系统市场取得了重大突破。

在个人机刚刚进入全盛时期的1980年左右，操作系统设计者认为 Unix 和其它传统的分时系统笨重、麻烦、不适合单用户个人机这个美丽新世界，而弃之不理 ——根本不顾 GUI 接口往往要求改造多任务处理能力，来适应不同窗口及其部件的绑定执行线程的事实。青睐客户端操

作系统的趋势非常强烈，服务器操作系统就像已经逝去的蒸汽机时代的遗物一样遭到冷落。

但是，正如 BeOS 设计者们所注意到的那样，如果不实现某些近似通用分时系统的东西，就无法满足普遍联网的要求。单用户客户端操作系统在互联网世界里不可能繁荣。

这个问题促使客户端操作系统和服务器操作系统重新汇到了一起。首先，互联网时代之前的1980年代晚期，人们首次尝试通过局域网进行点对点联网，这种尝试暴露了客户端操作系统设计模式的不足：网络中的数据必须放到集合点上才能实现共享，因此如果没有服务器就做不到这一点。同时，人们对 Macintosh 和 Windows 客户端操作系统的体验也抬高了客户所能容忍的最低用户体验质量的门槛。

到了1990年，随着非 Unix 分时系统模型的实际消亡，客户端操作系统设计者还是拿不出来多少可能解决这一挑战的方案。他们可以吸收 Unix（如 MacOS X 所做的），或通过一次一个补丁重复发明一些大致等价的功能（如 Windows），或试图重新发明整个世界（如 BeOS，但失败了）。但与此同时，各种开源 Unix 的类客户端能力不断增强，开始能够使用 GUI 并能在廉价的个人机上运行。

然而，这些压力在两类操作系统上并未达到上面描述所意味的那种对称。将服务器操作系统特性，如多用户优先权组和完全多任务处理，改装到客户端操作系统上非常困难，很可能打破对旧版本客户端的兼容性，而且通常做出的系统既脆弱又令人不满意，不稳定也不安全。另一方面，将 GUI 应用于服务器操作系统，所出现的问题却大部分可通过灵活处理和投入更廉价硬件资源得到解决。就像造房子一样，在坚实的地基上修理上层建筑当然要比更换地基而不破坏上层建筑来得容易。

第 8 章 磁盘 分区

目录

分区概念

首先我们需要知道，硬盘分区的存在，是由硬盘的物理特性决定的，并不会因为操作系统的不同而有所改变。

请您把一块硬盘想象为一本书......即便您不喜欢读书，您也一定非常熟悉它，所有的书都是相同的，包括我们使用的课本......您肯定非常熟悉

一本完整的书，通常包括书名、目录和正文。

如果您需要Linux，您首先需要找到一本书名为《linux》的书，书名相当于硬盘中的MBR，也就是主引导纪录。不同的是，MBR可以是几个书名合在一起，类似于《XX合订本》。这部分内容暂时还没有什么实用价值，您只需要大概的了解。

而正文，就是硬盘中纪录的数据，这也非常容易理解，且对于安装系统并没有什么影响，所以现在我们来了解目录:

目录相当于硬盘中的分区表，书中的每一个章节，相当于硬盘中的一个分区，它起始和结束的页次，都可以在目录中找到。试想，如果阅读一本撕掉目录的书，您将很难找到您想阅读的部分。同样，如果没有分区表，操作系统也不能够在硬盘上定位数据的位置。

由于历史的原因，硬盘中的分区表大小受到了限制，最多只可以容纳四个分区（主分区）。如果一本书，它的目录最多只能有四个章节，那不是太可怕了么？很多书的内容远远不止四个章节啊！

于是聪明的人们想到了一个变通的办法，就是利用其中的一个章节，来存储其它部分的目录。比如第一章是前言，第二章是其它部分的目录，我们翻到第二章，呵呵，这里是第二个目录，因为只有第一个目录受四个章节的限制，所以这个目录的内容可以非常的详尽。第二个目录就是分区表中的扩展分区了，其中定义的章节，就是硬盘中的逻辑分区，不是很难理解吧？

明白了这一点，我们来看看Linux和Windows对于分区不同的表示方法：

可能您已经很熟悉Windows了，它使用盘符来表示分区，比如 C: D: E: ，每一个分区使用一个盘符来标识，而且顺序可以颠倒， D: 并不一定就是您系统中的第二个分区。（如果您给第二个分区分配最后一个硬盘盘符，把所有的盘符按顺序排列好，并且重装一次系统，您就会

理解什么叫作“头疼”了）

而在Linux中，分区是这样表示的

```
/dev/hda
/dev/hda1
/dev/hda2
/dev/hda5
/dev/sdb1
```

以 */dev/hda5* 为例*:*

因为在Linux中，每一个设备都是用 /dev/ 文件夹下的一个文件来表示，所以 /dev/hda5 中，/dev/ 表示的是根目录下的dev目录，我们来看剩下的部分 hda5 。

前两位的字母 hd 表示这是一块IDE硬盘，如果是 sd ，则代表SATA硬盘，或者闪存等外设。

第三位的字母 a 表示这是该类型接口上的第一个设备。同理， b、c、d...... 分别代表该类型接口上的第二三四......个设备。例如 hdc 表示第二个IDE接口上的主硬盘（每个IDE接口上允许一个主设备和一个从设备）。

第四位的数字 5 ，并不表示这是该硬盘中的第5个分区，而是第一个逻辑分区。因为在Linux中，为了避免不必要的混乱，分区的顺序是不能改变的，分区标识则由它们在硬盘中的位置决定。系统又要为所有可能的主分区预留标识，所以 1-4 一定不会是逻辑分区， 5 则是第一个逻辑分区，以此类推。

挂载分区

在Linux系统的安装过程中，您需要选择系统目录的挂载点。

我们知道，安装Windows时，我们可以选择把系统安装在哪一个分区，把系统挂载到分区上。而在Linux中则相反，我们要把分区挂载到系统中。

当我们使用Windows的安装方式，把系统挂载到分区上，我们就不可能把Windows目录放在C盘，而把MyDocuments目录放到其它分区。您或者出于习惯，或者出于数据安全方面的考虑，通常把文档放到其它分区中。但是Windows下很多软件保存文件的默认目录就是MyDocument 目录，这就比较不方便。

在系统安装完成后，我们还是可以将MyDocuments目录转移到其它分区中，不过有点麻烦，可能许多朋友还不知道怎么去做......

而任何一种Linux系统时，我们可以在系统安装时就把分区挂载到目录下， /home 目录相当于Windows的 MyDocuments ，我们可以把 /dev/hda5 挂载到此目录下，这样我们往 /home 目录里存东西的时候，其实保存在第一个扩展分区中。如果再一次安装系统，只要把这个分区挂载到 /home 目录下，那么进入新系统就像回家一样，眞是太棒了。

理论上来讲，您可以将分区挂载到任何目录下面，您可以自定义挂载的路径。但是我们并不推荐您这么作，因为那没有任何意义。

系统安装程序向您建议的挂载目录，通常也是我们向您建议的，现在我们来了解一下，这些目录通常都是用来作什么的：

/

根目录，唯一必须挂载的目录。不要有任何的犹豫，选一个分区，挂载它！（在绝大多数情况下，有2G的容量应该是够用了。当然了，很多东西都是多多益善的）

swap

交换分区，可能不是必须的，不过按照传统，并且照顾到您的安全感，还是挂载它吧。它的容量只要大于您的物理内存就可以了，如果超过了您物理内存两倍的容量，那绝对是一种浪费。

/home

前面已经介绍过了，这是您的家目录，通常您自己创建的文件，都保存在这里，您最好给它分配一个分区

/usr

应用程序目录。大部分的软件都安装在这里。如果您计划安装许多软件，建议您也给它分配一个分区

/var

如果您要作一些服务器方面的应用，可以考虑给它分配一个较大的分区

/boot

如果您的硬盘不支持LBA模式（我想那不太可能:），您最好挂载它，如果挂载硬盘的第一个分区，应该比较稳妥。一般来说，挂载的分区只要100M大小就足够了

在文件系统这一环节中，我们建议您选择： EXT4

> 提示：也许您注意到了，Windows中，盘符既用于表示硬件（硬盘上的分区）,又用于表示系统中的路径。而Linux中，硬件就是硬件，路径就是路径，不会混淆在一起，简单直接！

第 9 章 文件系统

第 10 章 发行版介绍

第 11 章 编程语言

第 12 章 无根的根：无名师的 Unix 心传

目录

> 注意：《Rootless Root:The Unix Koans of Master Foo》为《Unix编程艺术》一书的附录，原作者不详

导言

《无根的根》这部心传的收录集在西山纯净的空气中得以保存数十年，它的发现在学术圈中掀起轩然大波。这些出土文稿是否为早期 Unix 教义的新发现？抑或仅是后世高明的赝品？那些半神秘的人物，像尊者 Thompson、Ritchie 和 McIlroy，是否以此发展出我们所处时代的教义？

答案无法确知。各方争论均被收入那本经典之作，《编程之道》(The Tao of Programming)。但是《无根的根》在论调风格上都与那本 James 松散诗化的逸闻译作有显著差异，所有一切都围绕着卓越而谜一般的无名师。

把 AI Koans(AI 心传)与之相提并论颇为恰当；在 AI Koans 中也可发现《无根的根》作者的着笔痕迹。它和 Loginataka(箴言剧)也有密切关联；实际上，《无根的根》和 Loginataka 大有可能是出自一人之手笔，此人正是无名师。

Tales of Zen Master Greg(禅祖 Greg 传说)也值得一说，对“九寸钉”的引用已经使人对其古老程度产生怀疑，它对《无根的根》影响微乎其微。

毫无疑问，可以认为标题借鉴了 Mumon 的禅宗经典读物《无门之门》(The Gateless Gate)。数个心传都可以看作是对 Mumon 的回应。

无名师应该归于东派(New Jersey)还是西派(尊者 Thompson 往 Berkeley 的划时代之旅)。如果不能回答这个问题，我们甚至也许无法宣称无名师确实存在，它也许只是一群教师，也许是一批达摩尊者。

即使把无名师的传说附会在单独的人身上，那他最看重的学生 Nubi 怎么办？ Nubi 是个有血有肉的形象，是个完美的门徒。也许有人会想起佛祖最喜爱的追随者阿难达的传说，但他的性格特质在神话传说中并没留下痕迹，而佛祖也是永恒的谜。

最终，我们可做的便是讲述故事的本原，抽丝剥茧，挖掘其中眞意。

《无根的根》还在编写之中，素材的整理和解释都是难题。难题解开后，将被收录到未来版本中。

无名师与万行码

无名师曾对来访的程序员说：“Unix传统上认为，一行shell脚本胜过万行C程序。”

这个程序员自以为对C极其精通，说：“这不可能。UNIX内核正是用C实现的。”

无名师回道：“确是如此。不过，UNIX传统上认为，一行shell脚本胜过万行C程序。“

程序员颇为沮丧：”但是在C中我们可领会到尊者Ritchie的智慧。我们与操作系统和机器合而为一，可以获取无与伦比的性能。”

无名师回道：“诚如你言。不过，Unix传统上认为，一行shell脚本胜过万行C程序。”

程序员冷笑着想愤然离去。无名师向学生Nubi颌首示意，Nubi在黑板上写下一行shell脚本，问道：“尊敬的程序员，看看这行管道，用纯C实现，是不是要一万行C代码？”

程序员沉吟念诵。最终他承认如此。

“你需要多长时间来实现和调试那个C程序？”Nubi问道。

“很长”，来访的程序员承认。“但傻子才会干这个而不去完成更有价值的任务。”

“那么谁更了解Unix传统？”无名师问道。“是写一万行代码的，还是看到任务的无谓而不去编码的？”

听到此，程序员眼中一亮。

无名师与脚本狂

无名师和学生吃早饭时，从黑客大陆来了个陌生访客。

“Ihear y00 are very l33t,”他说。“Pl33z teach m3 all y00 know”。（我听说你很牛，请把你会的都教给我。）

无名师的学生面面相觑，都没听懂这类粗鄙言语。无名师微笑道：“你想弄懂Unix？”

“I want to b3 a wizard hax0r”，陌生人回答，“and 0wn ever3one's b0xen。”（我想当个顶尖黑客，能掌握所有人的机器。）

“我不教这个”，无名师答道。

陌生人很激动。“D00d， y00 r nothing but a p0ser。”，他说。“If y00 n00 anything, y00 wud t33ch m3。”（哥们儿，敢情你没真本事啊，你要知道点儿东西就教给我了。）

“有条路，”无名师说，“可以将你带入真知。”他在纸上写了个IP地址。“黑掉这台机器，这对你来说应该不费什么力气，它的管理员不称职。回来后告诉我你发现了什么。“

陌生人鞠了一躬就离开了。无名师把他的早饭吃完。

几天过去了，几个月过去了。没人再想起陌生人。

数年过去了，黑客大陆来的陌生人回来了。

”你混蛋！“他说，”我黑掉了那台机器，你说的没错，太容易了。但是我被FBI抓起来扔进监狱了。“

”好“，无名师说，”你可以继续下一课了。“他在另一张纸上写了个IP地址交给陌生人。

”你疯了？“陌生人喊道。”经过这事，我再也不黑别人的机器了。”

无名师脸现微笑。“这里就是”，他说，“真知的开始。”

听到此，陌生人眼中一亮。

无名师的双路论

无名师如是教导学生：

“达摩教义有条准线，这在尊者McIlroy的符咒“做一件事并做好”中得到体现。它强调软件应当具有简单一致的行为，这符合Unix惯例，人和其它程序便都很容易想象其心理模型。

“但达摩教义还有另一条准线，体现在尊者Thompson的符咒“有怀疑，用穷举”中，很多经文都教导我们现在得到的90%，比等不来的100%更有价值。它强调实现的健壮性和简单性。

“现在告诉我：什么程序符合Unix传统？“

想了一会儿后，Nubi沉思道：

“老师，这些教义有冲突。”

“简单的实现往往对边缘情况有欠考虑，比如资源耗竭、无法关闭竞争窗口以及在未完成事务中超时等等。”

“发生边缘情况时，软件行为往往不规律、难以猜测。这当然不是Unix传统。“

无名师颔首同意。

“另一方面，大家都知道精巧的程序很脆弱。更进一步说，每个对边缘情况的修正往往牵扯到程序的核心算法，还牵扯处理其它边缘情况的代码。”

“于是，对边缘情况防患于未然、确保描述的简单性，反而会使得代码过分复杂、bug成堆、根本无法发售。这当然不是Unix传统。”

无名师颔首同意。

“那么，什么是正确的达摩道？”Nubi问道。

无名师说：

“当鹰飞翔时，它忘记爪子与地面相触？当虎捕食时，它忘记腾空的一刻？VAX只重三斤！”

听到此，Nubi眼中一亮。

无名师与方法论

无名师和学生Nubi在圣地行走，无名师习惯在晚间为城市和乡村的Unix新门徒布道。

一次，聆听者中混入了一名方法论者。

“优化程序时不对热点进行反复衡量，就像渔夫把网撒入空湖中。”无名师说。

“那么，”方法论者说，“管理资源时不持续地衡量你的产能，不也像渔夫将网撒入空湖中么？”

“我一次碰到一个渔夫时，他正将网撒入船下的湖中，”无名师说，“他摸了好一会儿船底，像在寻找他的船。”

“但是，”方法论者说，“如果他把网撒入湖中，为什么还要找船呢？”

“因为他不会游泳。”无名师答道。

听到此，方法论者眼中一亮。

无名师的GUI论

一晚，无名师和Nubi参加一个程序员的探讨会。有个程序员问Nubi和他的老师来自哪所学校。当得知他们是Unix大道的追随者时，程序员颇为不屑。

“Unix命令行工具太粗糙太落后”，他讥讽道。“现代的、设计得当的操作系统可以在图形用户界面中做任何事情。”

无名师一言不发，只是指着月亮。旁边的一条狗对着他的手狂吠。

“我不明白。”程序员说。

无名师依然缄默，指着一幅佛祖像，然后又指着一扇窗。

“你想说什么？”程序员问。

无名师指着程序员的头，接着指着一块大石。

“请把话说清楚！”程序员要求道。

无名师深深蹙眉，轻拍程序员的鼻子两下，把他扔到旁边的垃圾箱中。

程序员试图从垃圾堆挣扎出来之时，那条狗跑过来在他身上便溺。

此时，程序员眼中一亮。

无名师与Unix狂

一个Unix狂热者听说无名师掌握Unix大道真知，便跑来求教。无名师对他说：

“当尊者Thompson发明Unix时，他并不理解它。随后他理解了，受益了，不再发明了。“

“当尊者McIlroy发明管道时，他只知道它将传递软件，并不知道它能传递思想。”

“当尊者Ritchie发明C时，他将程序员放到缓冲溢出、堆损坏和烂指针bug的地狱中惩罚。”

“说实话，这些尊者又瞎又蠢！”

狂热者对无名师的用词极为愤怒。

“这些智者”，他抗议道，“给了我们Unix的大道。我们嘲笑他们，就是混淆是非，比转世为牲畜和MCSE还不如。”

“你的代码全无污点和缺陷？”无名师问。

“不，”狂热者承认，“没人不犯错。”

“这些尊者之智，”无名师说，“就是了解自身之愚。”

听到此，狂热者眼中一亮。

无名师的Unix传统论

一学生对无名师说：“我们听说SCO公司把握着纯正的Unix。”

无名师颔首。

学生继续说，“我们还听说OpenGroup公司也把握着纯正的Unix。”

无名师颔首。

“这怎么可能？”学生问。

无名师答道：

“SCO确实把握着Unix源码，但是Unix的源码不是Unix。OpenGroup确实把握着Unix的名称，但Unix的名称不是Unix。”

“那么，什么是Unix传统？”学生问。

无名师答道：

“非源码。非名称。非思想。非实物。恒变。不变。”

“Unix传统是简单和空。正是简单，正是空，才使得它更强胜飓风。”

“以自然法则前行，在程序员手中，吸纳各种优良设计。与之竞争的软件最终必与之想像；空，空，眞空，虚无，万岁！”

听到此，学生眼中一亮。

无名师与最终用户

无名师又一次布道时，一个最终用户听说了他的智慧，跑来求教。

他对无名师三鞠躬。“我欲学习Unix大道，”他说，“但是弄不懂命令行。”

一个旁观的新门徒开始嘲讽最终用户，说他脑子一锅粥，说只有经训练者、有智慧者才配使用Unix。

无名师抚手不语，命这个嘲笑最终用户的新门徒前坐，坐到最终用户身边。

“告诉我，”他对新门徒说，“你写过什么代码，有过什么突出设计。”

新门徒嗫嚅了两句，然后沉默了。

无名师转向最终用户。“告诉我”，他问，“为何你要寻求大道？”

“我用的软件并不能令我满意”，最终用户答，“既不稳定，也不美观。听说Unix之道尽管艰难，但超越一切，我愿抛去一切诱饵和虚像。”

“那么，”无名师问，“你为何想尽办法让软件帮你做事？”

“我是个建筑工”，最终用户答道，“这座城里的很多房屋都出自我手。”

无名师转向新门徒。“家猫也能欺负老虎”，无名师说，“但是猫叫永远比不过虎吼。”

听到此，新门徒眼中一亮。

部分 II. 地理

目录

第 13 章 基础知识

目录

Shell

可能您早已能够熟练的使用 GUI（图形用户界面），例如您可以使用鼠标双击一个图标，来打开或者执行它。

我们来看这个过程: 您使用鼠标定位桌面上的一个程序图标，按下左键两次。系统读取鼠标指针的位置，并且判断该位置下图标的涵义，根据预设的双击动作，运行程序或者打开文件。

这一套 GUI 系统，便是一种 Shell，它的作用是实现人机交互。如果我们不能够控制电脑，那么电脑还不如电视机好玩，不是么？电视机也可以选择频道（电视机的遥控器，也是一种人机交互的界面，不过相对于电脑，确实是相当简单了）

易于上手、界面直观是 GUI 的优点，但是 GUI 并不意味着简单！或许您有类似经历: 桌面上有几十个程序的启动图标，也知道它们的名字，但是翻出一个来，并不是一件轻松的事情。

我的 Windows 系统中，桌面上摆满了各种图标，每当启动一个程序的时候，我都很是困扰。后来尝试了 音速启动 这类程序启动管理器，效果却不遂人意。[18]

在 Linux 下，所有的程序都可以通过命令运行。虽然 Linux 也有 GUI，但是它并不比 Windows 的 GUI 更优秀！上面只是简单的介绍了 CLI（命令行界面）相对 GUI 的优越之处，使用 CLI 还有更多的好处，您会慢慢体会到的。

当然了，在您的印象中，CLI 一定非常的不友善，缺少亲和力，冷漠而拒人于千里之外......您和 CLI 之间甚至有代沟的存在

> [18] 在我的不懈努力下，这个难道最终得到了解决：将快捷方式名称简化，放到特定目录下，使用 Win+R 组合键呼出 运行 对话框，键入快捷方式的名称来运行该程序。比如 反恐精英 的快捷方式为 cs ，我把它放在 Windows 目录下; 运行 cs 命令，就可以去维护世界和平了。
>
> 这么多快捷方式，统统放到 Windows 目录下，非常混乱。因此，我在 D盘 建立了一个名为 path 的目录，并把它的路径加入到环境变量的 path 项中，快捷方式放在 D:\path 目录中。即便重装系统，只要在环境变量中重新加入此路径，原来的程序大多可以直接以命令来运行......我的许多朋友强烈要求我帮他们设定这种启动方式，因为这确实很方便：）

命令

坦白的说，冷不丁见到那么老长的一串命令，谁都会毛骨悚然。

也许您使用过 DOS，留下这种印象：命令先放一边，光是正确的输入目录、文件名都够瞧的。而且 DOS 不区分大小写，要是像 Linux 一样区分大小写，那多恐怖啊！！！

其实 Linux 命令行具有补全功能，非常实用。假设有这样一个命令：

```
command path/file
```

如果只有一个以 `c` 起始的命令，键入 `c` ，再按一次 `tab` 键，系统将自动补全该命令余下的部分。 只要 `c` `tab` 两次按键，就可以完成 **command** 的输入。

现在我们来了解命令的语法结构，这一部分相当重要，您可得看仔细。

我们知道，任何语言都有特定的语法结构，以我们的中文为例：

```
我们郑重地推荐您Linux！
```

这个句子的语法尽管简单，却是大部分的命令行采用的句型。让我们看一下，这个句子里都有些什么：

> ❶ 主语，Linux 命令的执行者当然是用户，所以主语一概省略。
>
> ❸ 一个动词，作为谓语而存在。Linux 命令中，这一部分是必须的。这一部分也是不同命令之间最根本的区别方式，所以它通常作为命令名，写在最前面。键入 date 命令，您可以查看当前的时间日期。
>
> ❷ 状语，用来修饰谓语。与之相对应，Linux 命令可以使用选项来精细调节程序的行为。为了与命令的操作对象相区别，选项前通常要加 - 或者 -- 符号。原则上，在命令名之后，选项的位置可以随意，但是为了养成一个良好的习惯，我们建议您在命令名后直接跟选项。
>
> ❹ 这两个部分都是宾语，它们是命令的操作对象。大部分的命令只有一个操作对象，也有一些命令是双宾语结构的，具有一个直接宾语和一个间接宾语。比如 拷贝 这个命令 cp (copy)[19]

分隔符. 我们的汉语是象形文字，没有分隔符。但是所有的拼音文字中都用空格作为分隔符，来分隔单词。Linux 命令中同样使用空格作分隔符。

上面的那句话，翻译成 Linux 的命令，应该是这个样子的：

```
推荐  --郑重的 您 Linux
```

> ❶ （按照传统，“-”后跟选项缩写，“--”后跟选项全称。不过也有例外）

哈，Linux 的命令也蛮简单吧？

> [19] **cp a b** 表示把文件 a 拷贝为文件 b
>
> **cp a /home**表示把当前目录下的 a 文件，拷贝到 /home 目录下。（命令的不同部分使用空格分隔，连续的空格视为一个空格）

Linux 程序、进程

或许您会这样想，Linux 命令的句型确实不难，但是那么多命令，我怎么知道它们都是作什么的呢？而且不同的系统中，可以使用的命令似乎也不太一样，这真让人困惑......

其实 Linux 的命令，运行的是 Linux 系统中的程序。只要您已安装了程序，您就可以通过命令来运行它，并且可以使用选项来精细的调整它的运行状态。也可以通过点击启动图标来运行，不过启动图标不能够方便的调整选项，并不是很方便。

举一个例子：

```
mplayer -shuffle -loop 3 -playlist mymp3.list
```

可能您运行上面命令，系统会提示您 无法找到命令 ，那是因为您没有安装 mplayer 这个程序。mplayer 是我见过的支持格式最多的播放器，几乎任何已知格式的多媒体文件，都可以使用 mplayer 来播放。它包含一个图形界面的前端，您可以在菜单中找到它，鼠标点击运行;也可以通过执行命令来运行它的命令行版本。

如果您的系统中没有 mplayer 播放器，我们建议您安装一个。关于程序的安装，参阅第 16 章 软件管理 。

上面命令中， **mplayer** 调用了 mplayer 播放器程序。选项 **-shuffle** 表示随机播放， **-loop** 表示循环播放，后面的 **3** 为循环的次数，如果为 **0** ,则一直播放。 **-playlist** 表示播放列表中的曲目。我们可以把 mp3 的路径放到 `mymp3.list` 文件中，让 mplayer 来播放它们。

进程. 为运行中的程序，是程序在内存中的镜像。

好了，现在您已经了解了 Shell 、 命令 、 程序 、 进程 的概念，您基本上也就了解了 Linux（Linux 系统真是非常简洁，而且容易理解）。

但只知道这些，您并不能顺畅使用。接下来的章节中，我们来进一步介绍它的细节。

路径

路径分为绝对路径和相对路径。

绝对路径的起始点为根目录 `/` ，例如 /usr/local/bin 就是绝对路径，它指向系统中一个绝对的位置，不受其它因素影响。

相对路径的起始点为当前目录，如果您现在位于 /usr 目录，那么相对路径 local/bin 所指示的位置为 /usr/local/bin

也就是说，相对路径所指示的位置，除了相对路径本身，还受到当前位置的影响。例如 Linux 系统中常见的目录 /bin 、 /usr/bin 、 /usr/local/bin ，如果只有一个相对路径 bin，那么它指示的位置可能上面三个目录中的任意一个，也可能是其它目录。

如果我告诉您到 bin 目录寻找一个文件，您可能搞不清楚是哪一个 bin 目录。只有当前位置确定，相对路径指示的位置才能够确定。

现在我说， /usr/local 目录下，它的相对路径 bin 中有某个文件，这样就比较明确了。

在相对路径中 . 表示当前目录， `..` 表示当前目录的上一级目录。

假设您安装了一个程序，它的主程序没有被放置到上面三个 bin 目录中的任何一个，或者其它系统能够找到的地方，您就得告诉系统，它的可执行文件在哪里。

可以使用绝对路径，例如： **/home/user/bin/**可执行文件

或者定位到 /home/user/bin 目录，使用相对目录来定位它 **./**可执行文件

路径相关命令

```
cd (change directory)        #更改目录。
pwd (print working directory)#显示当前路径。
ls (list)                    #显示当前目录中的文件列表。
```

请尝试以下操作：

```
cd /etc             #进入“/etc”目录，这里使用的是绝对路径
pwd                 #显示当前路径，这个命令返回结果“/etc”
cd init.d           #进入“/etc”目录的子目录“init.d”，这里使用的是相对路径
cd ..               #进入上一级目录“/etc”
cd ../home          #“/etc”目录的上一级目录为“/”，它的子目录“home”为“/home”
cd -                #回到上一次的目录，我们在“/etc”目录跳转到“/home”目录，所以这次是回到“/etc”目录
cd ~                #“~”代表当前用户的“$HOME”目录，也就是“/home/{用户名}”目录。
ls                  #在任何时候，您都可以使用“ls”命令，来了解当前目录下都有哪些文件。
```

远程路径：

远程路径的表示方法为 协议://用户名:密码@位置/路径:端口

大多数的远程路径可以使用默认端口匿名访问，由此用户名、密码、端口通常不需要填写。例如：

```
http://linuxtoy.org
```

要求身份验证的远程路径，您可以使用下面的方式访问:

```
ftp://user:passwd@ftp.linuxtoy.org:21
```

软件

Linux 中没有 注册表 这个概念。安装软件，理论上讲，只要拷贝所有相关文件，并运行它的主程序就可以了。

按照传统，一个软件通常分别拷贝到同级目录下的 bin、etc、lib、share等文件夹。

bin

可执行文件，程序的可执行文件通常在这个目录下。在环境变量中设定搜索路径，就可以直接执行，而不需要定位其路径。

etc

配置文件，大部分系统程序的配置文件保存于 /etc 目录，便于集中修改。

lib

库文件，集中在一起，方便共享给不同程序。相较不同的软件单独保存库文件，能够节约一些磁盘空间。

share

程序运行所需要的其它资源，例如图标、文本。这部分文件是专有的，不需要共享；而且目录结构相对复杂，混放在一起比较混乱，所以单独存放。

还有一些软件，占用一个单独的目录，所有的资源都在这个目录中。类似于 Windows 下的绿色软件，不推荐在 Linux 系统下这样作。

- 执行时，系统找不到可执行文件（搜索所有路径，资源开销过大，是不现实的），需要定位其位置，像这样 **/home/user/bin/**可执行文件 ，不够方便。
- 许多系统软件需要协作运行，配置文件分别保存，定位它们非常麻烦
- 如果程序使用的库文件，像图形库文件，都单独存放，那么磁盘空间的浪费会非常严重。

有一些大型软件，或者您布署的重要应用，您可以将它们单独安装在一个文件夹下。（通常源码安装支持这种方式)

配置方式

Linux 下没有类似 注册表 的系统，系统和软件都可以通过纯文本的配置文件进行设置。

事实上，图形界面的配置工具，通常就是以图形界面的方式修改配置文件，适合设置一些比较简单的程序。如果软件有几千个可以配置的选项，全部作成菜单，想象一下……开始发抖吧……

图形界面的配置工具，可以看作特定配置文件专用编辑器。您一样可以使用通用文本编辑器来编辑配置文件，比如 Nano、Gedit、Knote、Vim 或者 Emacs 等等。不考虑阅读、修改配置文本占用的时间，直接修改配置文件甚至更迅速。[20]

如果只是要修改某一常用选项，而且时常修改，比如主机的 IP地址。使用文本编辑器，您要找到相应的配置文件，还要在配置文件中找到相应的选项，会浪费掉您的时间和耐性。

图形配置工具经常会受各种因素制约，比如网络服务器中不提供图形服务，图形界面不够稳定……这时，您可以使用命令行的配置工具来完成这些工作。

例如： 修改主机 IP地址，可以使用 ifconfig 这个程序，执行下面的命令：

```
ifconfig eth0 192.168.0.1
```

> [20] 在以后的章节中，如果我们提示您修改某一文件，例如 `/etc/fstab` ，您可以使用任何顺手的文本编辑器打开它。

隐藏文件

Linux 下，名称中第一个字符为 . 的文件或者文件夹，系统会将它们隐藏起来。传统上，这种文件大多是用户的配置文件。

您可以尝试以下操作：

```
cd ~      #进入您的用户目录
ls        #查看当前目录下的文件列表
ls -a     #查看所有文件的文件列表（包括隐藏文件）。
```

- 如果您只想查看隐藏文件，而不包括这两个特殊目录，您可以使用 ls 命令的选项 -A （ls -A）
- 每个目录下都包含两个特殊目录 `.` 和 `..` 。您也许猜到了， `.` 代表当前目录， `..` 代表上一级目录。

现在，您可以看到许多文件名以 . 起始的文件或者文件夹了吧？使用 ls 命令无法显示它们

文件类型

Linux 系统主要根据文件头信息来判断文件类型，扩展名并非决定因素。

现在使用 **ls -l** 命令，查看详细信息格式的文件列表，您将会看到如下内容：

```
total 5
drwxr-x---   4 user  group   4096 Mar 10 00:37 filename
drwxr-xr-x 21 user  group   4096 Mar 10 20:16 文件名
-rw-------   1 user  group    524 Mar 10 00:40 a
-rw-r--r--   1 user  group     24 Jun 11  2000 b
drwx------   2 user  group   4096 Mar  9 11:06 c
```

共显示了七列信息，从左至右依次为：权限、文件数、归属用户、归属群组、文件大小、创建日期、文件名称

```
drwxr-xr-x
```

> ❶ 其中要特别留意的是第一列：

-	普通文件
d	文件夹
l	符号链接
b	块设备文件
c	字符设备文件
s	套接文件
p	管道文件

> ❷ 剩下的9个字符分为3组：分别是归属用户、归属群组、其它用户对于该文件的权限。我们看它的格式

r	可读
w	可写
x	可执行

- 它们的顺序不能颠倒，某一位置为空(-)，则表示不具有相应的权限。
- Linux 下的可执行文件并不是由扩展名（例如 .exe ）决定的，而是由其可执行权限位决定。
- 目录是一种特殊类型的文件。拥有目录的执行权限才可以进入这个目录!

权限

我们已经知道了，文件的权限分为 r （可读）、 w （可写）、 x （可执行）三种类型，而一个文件可以针对归属用户，归属群组，其它用户或群组分别设定权限。

这种权限管理的方式灵活、简单、严密、明晰。尽管如此，在最初的阶段，可能会有一点小小的不适。因为它无所不在，而您习惯了的 Windows 的权限管理却不是这样（非常混乱，大多数时间形同虚设，偶尔用到却让人伤透脑筋）。

使用 chmod 命令更改文件的权限，使用 chown 来更改文件的归属。例如：

```
chmod 755 xxx
chmod a+x xxx
chown user:group xxx   #用来更改文件的归属用户，也可以同时更改其归属群组
chgrp   group   xxx    #用来更改文件的归属群组
```

上面命令中的 755 和 a+x 是两种类型的表达式

我们将在“权限管理”一节中详细介绍

执行命令的权限

有一些命令，普通用户也可以执行，但是只有 root用户 才能执行成功，这是为什么呢？

例如在系统中增加一个新用户 useradd，我们看看这个命令的程序文件

```
ls -l /usr/sbin/useradd

-rwxr-xr-x 1 root root 56156 2006-04-03 21:37 /usr/sbin/useradd
```

所有的用户都可以执行？

这是因为， useradd 命令是修改 `/etc/passwd` 文件的一个工具，来看看这个文件：

```
ls -l /etc/passwd

-rw-r--r-- 1 root root 1835 2006-06-24 17:58 /etc/passwd
```

原来只有 root用户 才能写入修改结果，非 root用户 执行 useradd 命令当然不会有结果。

用户

在 Linux 系统中，有两种用户：普通用户、root用户

root用户拥有对系统的完全控制权。实际上这没有看上去那样的美妙，你必须对自己的行为负全部的责任。Linux 继承了 Unix 的设计哲学：系统毫无保留的执行你下达的命令，哪怕这个命令是“向我开炮！”

普通用户只可以作系统允许的事情。尽管可以执行大多数命令，但是 root 专有的命令却不能成功的执行，因为这些命令往往关联着只有 root 才可以处理的文件。也就是说，普通用户通常只可以处理自己 `$HOME` 目录下的文件。详见“执行命令的权限”一节

而作为 root用户，可能会因为误操作给系统带来破坏；作为普通用户，可以作的事情又太少了点。

执行命令的身份

默认情况下，您的命令提示符末位为 *$* ，这表示您将以普通用户的身份执行命令。

您可以使用 su （switch user）这个命令来切换其它用户。

例如 **su root** ，切换到 root用户(如果 su 命令后面没有参数，那么这个命令默认切换到 root用户)。

当您执行 su 这个命令，系统会提示您输入密码，请输入管理员的密码。这个时候，您会发现命令提示符末位变成了 **#** ，您将以 root用户 的身份执行命令。

许多 Linux 系统默认会随机设定系统的 root密码，这样更安全一些，这个时候您可以执行 sudo 命令，输入当前用户密码后，暂时以 root用户 的身份执行命令。（前提是 sudoer 列表中要包含您的 ID）

> 提示：使用 visudo (需要 root权限)可以将普通用户加入到 sudoer 列表中。

但即便是用 sudo 来执行，也不能保证系统不会在您的误操作下一命呜乎，更安全的办法是直接赋与用户某种操作的权限。

```
gpasswd -a user audio
```

这个命令将 user 加入到 audio 群组，您拥有了使用音频设备的权限。

事实上，Linux 中一切皆文件，包括设备文件。

```
ls -l /dev/sound

crw-rw---- 1 root audio 14,  4 10-04 09:41 audio
crw-rw---- 1 root audio 14,  3 10-04 09:41 dsp
crw-rw---- 1 root audio 14,  2 10-04 09:41 midi
crw-rw---- 1 root audio 14,  0 10-04 09:41 mixer
…………
```

可以看到，和音频有关的设备文件，除了 root用户 可以使用，audio群组 中的用户也可以使用。

群组

群组分为两种，主组和辅组。在 `/etc/passwd` 文件中，与用户相关联的是主组，每个用户只可以加入一个主组；`/etc/group` 文件中则记录了每一个辅组所包含的用户，同一个用户可以被多个辅组所包含。

这样看起来有点乱，但却可以提供更大的灵活性。例如：

```
crw-rw---- 1 root audio    /dev/sound/audio
brw-rw---- 1 root optical  /dev/cdrom
```

先假设一名用户只可以属于一个群组：你是系统管理员，你管理的用户要求听点音乐，你把他加入了 audio组；于是该用户把 CD 放进了光驱，结果一点反映也没有，于是为了让他使用 cdrom，不得不让他拥有 root权限，你将不能防范他可能作出的破坏。

而一名用户可以属于多个群组，能够很好的解决这个问题：你可以让可以用户既属于 audio群组，又属于 optical群组，这样他就可以用 cdrom 来听音乐,却没有 root权限。

第 14 章 命令系统

目录

Shell、Console、Terminal

在前面的章节中，我们曾提到，电视机的遥控器，也是一种人机交互的界面，算是一种 Shell。

但是这个概念并不准确，遥控器只是向 Shell 发送指令的工具，Shell 接收到遥控器发出指令后，将指令转换为系统命令，由系统来执行。

例如我们按的遥控器上的 数字键1 ,遥控器将 切换为1频道 的指令发送到 Shell，Shell 将指令转换为系统可以识别的 频道1 ，系统执行它，您就可以观看 1频道 的电视节目了。

通常每台电视机只有一种 Shell，比如有的电视机系统具有“画中画”的功能，那么 Shell 中便有相应的功能定义，您可以通过遥控器上的“画中画 ”功能键来开启它。假设您的电视机没有此功能，Shell 中也就没有相应的功能定义。拥有一个带“画中画”功能控制键的遥控器，即便信号兼容，您还是不能够使用这一功能

不用遥控器也可以控制电视机，假设您的遥控器丢了，您还可以走到电视机前，使用机身上的控制面板来控制它（相当于使用 Linux 的控制台）。但是您一定不喜欢这种方式，除非您想锻炼身体

在 Linux 系统中，由于图形界面和控制台的分辨率通常不一致，所以切换时要有一个延时。对于我们中文用户来讲，控制台下中文的显示也比较麻烦。而且控制台显示内容通常不如终端显示的全面。

所以我们推荐您使用终端来执行命令，它使用起来感觉很像遥控器

rxvt-unicode

通常情况下，您买一台电视机，只能获得一个遥控器。虽然它为您的电视机量身定作，能够最大限度发挥电视机的能力，但您却不一定喜欢它。说不定这个遥控器体形太大，持握不方便;或者它体形太小，容易失踪;又或者它的按键要么太硬，要么太软;它的键盘要么太大，要么太小……

您一般也可以容忍，毕竟遥控器使用频率并不算高

如果您的终端有些地方不讨您喜欢，比如说响应太慢，或者不能正常显示中文……那就难以忍受了，您应该换一个其它的试试。

您所接触的第一个终端极有可能是 Gnome-Termianl，它是系统默认使用的终端，显示中文不错，不过响应比较慢。。。

我们推荐您使用 urxvt（mlterm 也是不错的选择）

您可以使用 **sudo apt-get install rxvt-unicode** 命令来安装它。

`urxvt` 启动它 （urxvt 不支持控制台，您得在图形界面下启动它。使用终端，或者按下 Alt+F2，建议您在启动栏里新建一个启动图标）

rxvt-unicode 还支持“服务器/客户端”的运行模式：

- `urxvtd` 启动一个守护进程 daemon（支持控制台）
- `urxvtc` 启动客户端 client。多个客户端可以同时连接到一个 urxvtd，以达到节省系统资源的目的。

或许您对 rxvt 的默认设置不满意，您可以修改用户配置文件 `~/.Xresources` 来设定它。修改全局配置文件 `/etc/X11/Xresources/Xresources` ，则对所有用户生效，只有 root 才可以修改此文件。

这里有一些简单的选项：（以 ! 起始的行是注释，您可以直接拷贝此文件的内容）

例 **14.1. urxvt** 配置 `~/.Xresources`

```
!!=========================================================================
!! RXVT-unicode setting
!!=========================================================================
!设置字体分辨率
Xft.dpi:96
!设置字体
URxvt.font:-misc-fixed-medium-r-normal--14-*-*-*-*-*-iso10646-1,xft:WenQuanYi Bitmap Song
!颜色
Rxvt.background:black
Rxvt.foreground:white
Rxvt.colorBD:yellow
Rxvt.colorUL:green
!滚动条
Rxvt.scrollBar:True
Rxvt.scrollBar_right:True
Rxvt.scrollBar_floating: False
Rxvt.scrollstyle:plain
!屏幕缓冲
Rxvt.saveLines:10000
Rxvt.color12:DodgerBlue
Rxvt.menu:/etc/X11/rxvt.menu
Rxvt.preeditType:Root
!输入法设置
!inputMethod:xim
```

帮助系统

您可以使用命令 man 或者 info 来阅读 Linux 命令的在线文档。命令的格式非常简单：

```
    man xxx
```

> 提示：在使用 man 浏览器的时候，一些快捷键您可能会用到：

Ctrl+f(orward)	向下翻一页	Ctrl+d(own)	向下翻半页
Ctrl+b(ackward)	向上翻一页	Ctrl+u(p)	向上翻半页
/	查找	q(uit)	退出

以上为 VI风格 的键绑定。您也可以使用 Emacs风格 的键绑定

bash

好了，现在我们换了一个遥控器，感觉顺手多了。现在来操练一下，下载一首 mp3：

我们使用 wget 这个程序，它非常可靠，完全值得您信赖。

首先找到一个可以下载的地址，复制链接，在终端窗口内点击鼠标中键，把它粘贴进去。

现在终端中大概是这种情形：

```
http://linuxtoy.org/xxx.mp3
```

按下 `Ctrl+a` 组合键，我们发现光标移动到了行首。输入 `wget` 和 `空格`

```
wget http://linuxtoy.org/xxx.mp3
```

回车后，终端中出现一些信息，不一会儿工夫，mp3 便下载完成。

使用 `Ctrl+a` 组合键，我们就不需要使用方向键来移动光标，方向键每次只能移动一个字符，没有效率

您还可以使用 `Ctrl+f` 向前移动光标， `Ctrl+b` 向后移动光标， `Ctrl+e` 将光标移动到行末……（键绑定）

> 注意：Linux 的图形界面中，鼠标中键通常执行“粘贴”的操作，如果您的鼠标没有中键，您可以左右键同时按下。

中止正在运行的程序

如果一个命令持续时间很长，以致于不能够进行其它操作，可以使用 `Ctrl+c` 来强行中止它。

Ctrl+s

有时您会不小心按下 `Ctrl+s` 这个组合键，Shell 便被冻结。尝试使用 `Ctrl+q` 组合键，看能否恢复正常。

键绑定

等等，有必要记这么多快捷键么？都这么复杂！

我们强烈建议您记住，以大幅度的提高操作效率。而且这是 readline 控件的键绑定，在任何使用 readline 控件的程序中，您都可以使用它们。例如 bash、lftp、gdb 等程序;同时，Linux 下倍受追捧的 Emacs 编辑器，也是这种风格的键绑定（其实是 readline 使用了 Emacs风格的键绑定才对），甚至 FireFox 中，也可以使用类似风格的快捷键！（Linux 下主要有两种风格的键绑定，一种是 Emacs风格，另一种是 Vi风格，我们会在 第 24 章 *Vim 编辑器* 中介绍）

现在列举一些 ReadLine 的键绑定，您可以自行尝试。（运行 `man readline` 命令，来查看 ReadLine 手册）

先来了解一些约定：

- \C-a 表示 `Ctrl+a`
- \M-a 表示 `Meta+a` Meta键 在 PC 中通常为 ALT键

表 14.1. Emacs风格 键绑定

	向前	向后		
移动				
行	\C-a	Ahead	\C-e	End
字符	\C-f	Forward	\C-b	Backward
单词	\M-f	\M-b		
命令	\C-n	Next	\C-p	Previous
删除				
行	\C-u	\C-k		
字符	\C-h	\C-d		
单词	\C-w	\M-d		
搜索				
搜索	\C-r (连续使用 C-r 可以查找下一个)			
浏览搜索结果	\M-n	\M-p		
其它				
粘贴	C-y			
撤销	C--			
清空屏幕	C-l			

自动补全

Tab	使用频率最高的功能！	
\C-o	遍历补全	（未定义）
\M-?	列出所有可能选项，相当于按两次 Tab键	Alt+Shift+/
\M-#	注释掉当前命令，用于将当前命令暂存于历史纪录列表	Alt+Shift+3
\M-!	补全命令，通常用来补全子命令，例如 `sudo` 的子命令	Alt+Shift+1
\M-~	补全用户名	Alt+Shift+`
\M-@	补全主机名	Alt+Shift+2
\M-$	补全变量	Alt+Shift+4
\M-_	补全历史纪录中的纪录	Alt+Shift+-
\M-*	将所有可能选项放到命令行中	Alt+Shift+8

自定义键绑定

通过修改 `/etc/inputrc` 文件，可以更改键绑定。建议您使用默认的键绑定，以避免不必要的烦恼。当然了，Emacs风格 的键绑定是通用的，随时都有可能用到。

在文件中添加该行，可以将 ReadLine 的键绑定设为 VI风格。（Bash、Lftp 等使用 ReadLine 的软件同时生效）

```
set editing-mode vi
```

找到这一行：

```
      $if mode=emacs
```

在它的下面添加如下内容

```
"\C-o": menu-complete

###这两行不是必须的，视情况而定###
"\C-p": non-incremental-reverse-search-history
"\C-n": non-incremental-forward-search-history
```

重新登录 Shell，您就可以使用 `\C-o` （Ctrl+o）来遍历补全。假如您的文件名为中文，或者出现乱码时，您可以使用 `\M-*` 将所有文件名放入命令行，再删除多余的，这眞是麻烦极了！简单一点的方法是使用 `\C-o` 遍历补全，将所有可能的选项轮流放入命令行。

> 提示：使用 Vim编辑器, **vi /etc/inputrc** ，在插入模式下使用 `Ctrl+v` 组合键。按下 `Ctrl+o` ，这时编辑区新增一个 ^O 字符，等价于 `\C-o`

通配符

使用 ? 代表任意单个字符。例如 ???lo ，表示 lo 前有三个字符，它可以匹配 Hello

使用 * 代表随意几个任意字符。例如 *.iso ，代表所有 iso 格式的文件。

您可以将遍历补全和通配符结合使用，以提高效率。例如：

```
cd */    则遍历补全只补全文件夹
chmview *.chm 则遍历补全只补全 chm 文件
```

任务管理

&

在命令的末尾加上一个 & 符号，表示背景任务，例如：

```
wget http://www.download.net/xxx/mp3 &
```

;

使用 ; 将多个命令连结起来，则表示任务按顺序执行

&&

使用 && 将多个命令连结起来，则表示只有前面的命令执行成功，后面的命令才能得以执行

``

`(命令)` ，如果一个命令中包含以 `` （Esc键 下方的按键）括起来的子命令，那么子命令将被优先执行，执行结果被代入上一级命令继续执行，例如创建一个以当前时间命名的文件：

```
touch `date +%m.%d_%H:%M:%S`
```

touch 命令能够创建一个文件，它的操作对象，为 date +%m%d%H%M%S 命令的输出 06.06_06:06:60

这样，我们创建了一个名为 06.06*06:06:60* 的文件（六月六日六时六分刚过六十秒--!）

Ctrl+z

将当前 Shell 中的任务挂起,这个时候任务的状态为

```
[1]+  Stopped   xxx
```

bg

将挂起的任务背景运行。这时它的状态为

```
[1]+ xxx &
```

fg

将背景任务调到前台执行

jobs

查看背景任务，方括号中的数字为命令的任务编号

如果后台运行多个任务，您可以在 bg 或者 fg 后跟任务编号，作为操作对象，例如：

```
bg 2
```

管道、重定向

>

重定向符号，它的作用是将命令的输出重定向到一个文件中。比如我们想把命令 **ls** 的结果保存为 `FileList` 文件，作一个清单，我们可以使用重定向符号来完成它：

```
ls -l &gt; FileList
```

>>

作用与 > 基本相同，不同点在于， >> 以追加的方式，将命令的输出写入文件的末尾。

<

是从文件到命令的重定向，将文件的内容作为命令的输入。

|

为管道符号，它的作用是将前一个命令的输出，作为下一个命令的输入。假设一个目录下的文件太多，使用 ls 命令不能够在屏幕中完全显示，这个时候您可以将 ls 命令的输出，通过管道符号，作为浏览器 less 的输入。就可以使用浏览器的功能翻页、查找：

```
ls -al | less
```

> 提示：less 浏览器的键绑定几乎与 man 相同，请参阅“帮助系统”一节

脱字符

Shell 中的一些功能是通过特殊符号作为控制字符来实现的，上面已经介绍了很多了。这产生一个问题，如果一个文件名中，刚好包含了这些字符，比如；，就很难对它进行操作。使用 less 浏览这个文件

```
less ;xxx
```

less 会很快返回一个错误信息，因为并没有一个文件名作为操作对象。接着，Shell 会报告，系统中没有 xxx 这个命令。

这是因为 Shell 将文件名中的 ; 解析为按顺序执行命令。

或者您的文件名以空白起始，而在 Shell 中，无论多少个空格，都将被解析为一个分隔符。您甚至不能使用命令重命名此文件。

这个时候就要用到脱字符 \ 了，它能够将一个具有特殊涵义的字符转换普通字符。上面的两个任务，可以在文件名中每个特殊字符前加一个 \ ，像这样

```
less \;xxx
less \ \xxx
less \;\ \&\xxx
```

> 提示：也可以使用 " 将文件名括起来，例如 **less "; &xxx"** ，在很多情况下，这样甚至更方便。

脱字符在 Shell 中也可以作为换行符，在一个命令的末尾添加一个 \ ，然后回车，在下一行继续输入命令剩余的部分，将一个命令拆分为多行且不影响它的执行（如果执行一个很长的命令，请将它拆分为多行以便于阅读）

事实上换行符也符合脱字符的定义。回车键有两个涵义，一个是 执行 （Enter），另一个 换行 （折线箭头）。在 Shell 中它作为控制字符 执行 ，使用脱字符后，它便代表排版字符 换行了。

设定您的默认 Shell

如果能够拥有 root权限，可以直接修改 `/etc/passwd` 文件。找到您 用户 ID 起始的行

```
user:x:1000:112:user,,,:/home/user:/bin/bash
```

> ❶ 用户登录名
>
> ❷ 用户口令（通常转储在 `/etc/shadow` 文件中）
>
> ❸ 用户UID
>
> ❹ 用户GID
>
> ❺ 用户信息
>
> ❻ 用户 `$HOME` 目录位置
>
> ❼ 最后一个字段为登录后的默认 Shell， `/bin/bash` 是程序 bash 的主程序路径。 Zsh 主程序的路径通常为 `/bin/zsh`

`/etc/shells` 中列出系统中所有可用 Shell（ `/bin/false` 代表禁用 Shell）

也可以使用如下命令更改您的默认 Shell

```
chsh -s /bin/zsh
（需要输入您的密码）
```

设定命令的搜索路径

使用 **echo $PATH** ，可以显示 `$PATH` 变量，输出如下：

```
/usr/local/sbin /usr/local/bin /usr/sbin /usr/bin /sbin /bin /usr/bin/X11 /usr/games /usr
```

它是一个环境变量，代表执行命令时，Shell 的搜索路径。

执行一个命令时，Shell 会到 `$PATH` 变量定义的路径去搜索，并运行与命令同名的可执行文件。如果程序、脚本等可执行文件并不在上面的路径中，就必须使用绝对路径或者相对路径定位可执行文件。

例如：

```
/usr/local/mplayer -menu xxx.rmvb

cd /usr/local/   &&  ./mplayer -menu xxx.rmvb
```

可以修改 `/etc/environment` 文件来设定您的命令搜索路径，找到 PATH 起始的行

```
PATH="$PATH:/user/"
```

在双引号中添加您的自定义路径，并以 *:* 分隔。

第 15 章 基本系统

目录

目录结构

各种 Linux 发行版的目录结构可能不太一样，但它们都遵循 FHS(Filesystem Hierarchy Stand)。

实际上 FHS 只是规定了根目录下的各主要目录应该放些什么文件，仅了解这些还不够，下面是一般情况下 Linux 系统的目录结构

```
/   根目录
│
├boot/                启动文件。所有与系统启动有关的文件都保存在这里
│    └grub/           Grub 引导器相关的文件
│
├dev/                 设备文件
├proc/                内核与进程镜像
│
├mnt/                 临时挂载
├media/               挂载媒体设备
│
├root/                root用户 的 $HOME 目录
├home/
│    ├user/           普通用户 的 $HOME 目录
│    └.../
│
├bin/                 系统程序
├sbin/                管理员系统程序
├lib/                 系统程序库文件
├etc/                 系统程序和大部分应用程序的全局配置文件
│   ├init.d/          SystemV 风格的启动脚本
│   ├rcX.d/           SystemV 启动脚本的链接，定义运行级别
│   ├rc.d/            BSD 风格的启动脚本
│   ├rc.xxx           BSD 风格启动脚本，定义运行级别
│   ├network/         网络配置文件
│   ├X11/             图形界面配置文件
│
├usr/
│   ├bin/             应用程序
│   ├sbin/            管理员应用程序
│   ├lib/             应用程序库文件
│   ├share/           应用程序资源文件
│   ├src/             应用程序源代码
│   ├local/
│   │     ├soft/      用户程序
│   │     └.../       通常使用单独文件夹
│   ├X11R6/           图形界面系统
│
├var/                 动态数据
│
├temp/                临时文件
```

启动流程

1. 读取 MBR 的信息，启动 Boot Manager❶
2. 加载系统内核，启动 init进程❷
3. init进程 读取 `/etc/inittab` 文件中的信息，并进入预设的运行级别，按顺序运行该运行级别对应文件夹下的脚本。脚本通常以 start 选项启动，并指向一个系统中的程序。❸
4. 根据 `/etc/rcS.d/` 文件夹中对应的脚本启动 Xwindow 服务器 xorg ❹
5. 启动登录管理器，等待用户登录 ❺

> ❶ Windows 使用 NTLDR 作为 Boot Manager，如果您的系统中安装多个版本的 Windows，您就需要在 NTLDR 中选择您要进入的系统。 Linux 通常使用功能强大，配置灵活的 GRUB 作为 Boot Manager，我们将在第 22 章 *Grub*中向您介绍它的使用方式。
>
> ❷ init进程 是 Linux 的根进程，所有的系统进程都是它的子进程。
>
> ❸ 通常情况下， `/etc/rcS.d/` 目录下的启动脚本首先被执行，然后是 `/etc/rcN.d/` 目录。例如您设定的运行级别为 3,那么它对应的启动目录为 `/etc/rc3.d/` 。
>
> ❹ Xwindow 为 Linux 下的图形用户界面系统。
>
> ❺ 大多 Linux 系统默认使用 GDM 作为登录管理器，您在登录管理器界面中输入用户名和密码后，便可以登录系统。（您可以在 `/etc/rc3.d/` 文件夹中找到一个名为 `S13gdm` 的链接）

更改运行级别

在 `/etc/inittab` 文件中找到如下内容：

```
# The default runlevel.
id:2:initdefault:
```

> ❶ 2 为系统的运行级别，默认的运行级别涵义如下：

0	关机
1	单用户维护模式
2~5	多用户模式
6	重启

服务管理

更改启动服务

在运行级别对应的文件夹中，您可以看到许多文件名以 S## 和 K## 起始的启动脚本链接。例如：

```
/etc/rcS.d/S35mountall.sh    挂载文件系统
/etc/rcS.d/S40networking     启用网络支持
/etc/rc2.d/S13gdm            启动登录管理器
/etc/rc2.d/S20makedev        创建设备文件
/etc/rc2.d/S23xinetd         启动超级进程
/etc/rc2.d/K20powernowd      针对某种硬件的电源管理支持
```

> ❶ init进程将以 **start** 为选项，按文件名顺序执行所有以 S## 起始的脚本。脚本名称中的数字越小，它将被越早执行。例如在 `/etc/rc2.d/` 文件夹中， `S13gdm` 文件名中的数字小于 `S23xinetd` ， `S13gdm` 将比 `S23xinetd` 先执行。
>
> ❷ 如果一个脚本链接，以 K## 起始，表示它将以 **stop** 选项被执行。如果相应服务没有启动，则不执行该脚本。

- 如果您想禁止某一服务在启动时自动运行，您可以将相应运行级别中的脚本由 S##xxx 重命名为 K##xxx 。

手动控制服务

您也可以手动运行带有以下选项的启动脚本，来控制系统服务。 **start** 启动 **stop** 停止 **restart** 重启

```
/etc/rc2.d/K20powernowd start
```

有时您并不清楚某一运行级别有什么启动脚本；而且此类脚本的前三位字符并不固定，不便于记忆。这时，可以直接使用 `/etc/init.d/` 文件夹中的启动脚本（ `/etc/rcX.d/` 中的启动脚本链接到 `/etc/init.d/` 文件夹下相应脚本），这也是推荐的方式。

```
/etc/init.d/powernowd start
```

> 注意：以上命令的位置并没有包含在环境变量的搜索路径中，所以要输入完整路径。

常见系统服务

acpi-support	高级电源管理支持
acpid	acpi 守护程序.这两个用于电源管理，非常重要
apmd	acpi 的扩展
alsa	声音子系统
alsa-utils	声音子系统实用工具
cron	任务调度系统，建议开启
anacron	cron 的子系统，将系统关闭期间的计划任务，在下一次系统运行时执行
atd	类似于 cron 的任务调度系统。建议关闭
binfmt-support	核心支持其他二进制的文件格式
bluez-utiles	蓝牙设备支持

bootlogd	启动日志。开启它
syslog-ng	系统日志，建议开启
klogd	同上，使用一种就可以了
sysklogd	
cupsys	打印机子系统
dbus	消息总线系统(message bus system)。非常重要
dns-clean	使用拨号连接时，清除 dns信息
evms	企业卷管理系统
fetchmail	邮件用户代理守护进程，用于收取邮件
gdm	gnome 登录管理器
gpm	终端中的鼠标支持
halt	别动它
hdparm	调整硬盘的脚本，配置文件为 `/etc/hdparm.conf`
hibernate	系统休眠
hotkey-setup	笔记本功能键支持
hotplug and hotplug-net	即插即用支持，比较复杂，建议不要动它
ifrename	网络接口重命名脚本。如果您有十块网卡，您应该开启它
inetd	配置文件为 `/etc/inetd.conf`
linux-restricted-modules-common	受限模块支持。❶
lvm	逻辑卷管理系统支持
makedev	创建设备文件，非常重要
mdamd	磁盘阵列
module-init-tools	从 `/etc/modules` 加载扩展模块，建议开启
networking	网络支持。按 `/etc/network/interfaces` 文件预设激活网络，非常重要
ntpdate	时间同步服务，建议关闭
pcmcia	pcmcia 设备支持
powernowd	移动 CPU 节能支持
ppp	拨号连接
ppp-dns	
readahead	预加载库文件

reboot	别动它
resolvconf	自动配置 DNS
rmnologin	清除 nologin
rsync	rsync 守护程序
sendsigs	在重启和关机期间发送信号
single	激活单用户模式
sshd	ssh服务器
sudo	检查 sudo 状态
udev	用户空间 dev文件系统（userspace dev filesystem）。重要
umountfs	卸载文件系统
urandom	随机数生成器
usplash	开机画面支持
vbesave	显卡 BIOS 配置工具。保存显卡的状态
xorg-common	设置 X 服务 ICE socket
adjtimex	调整核心时钟的工具
dirmngr	证书列表管理工具
hwtools	irqs 优化工具
libpam-devperm	系统崩溃之后，用于修理设备文件许可的守护程序
lm-sensors	板载传感器支持
mdadm-raid	磁盘阵列管理器
screen-cleanup	清除开机屏幕的脚本
xinetd	管理其他守护进程的一个 inetd 超级守护程序

> ❶ `/lib/linux-restricted-modules/` 文件夹中的模块为受限模块。例如某些驱动程序，如果您没有使用受限模块，就不需要开启它。

配置文件

> 小心：无论任何情况下，修改配置文件之前，先备份它

建议使用这个命令

```
sudo cp xxx xxx_`date +%y%m%d_%H:%M`
```

当然这很麻烦，您可以新建一个名为 bak 的文件，内容如下：

```
#!/bin/bash
sudo cp $1  $1_`date +%y%m%d_%H:%M`
```

把它放在您能够记住的目录下，比如 `/home` ，执行命令 **sh /home/bak xxx** ，就可以将当前文件夹下的文件 `xxx` 另存为 `xxx_yymmdd_HH:MM` 的格式了

全局配置文件

表 **15.1.**

系统初始化	/etc/inittab	运行级别、控制台数量
/etc/timezone	时区	
/etc/inetd.conf	超级进程	
文件系统	/etc/fstab	开机时挂载的文件系统
/etc/mtab	当前挂载的文件系统	
用户系统	/etc/passwd	用户信息
/etc/shadow	用户密码	
/etc/group	群组信息	
/etc/gshadow	群组密码	
/etc/sudoers	Sudoer 列表❶	
Shell	/etc/shell	可用 Shell 列表
/etc/inputrc	ReadLine 控件设定	
/etc/profile	用户首选项	
/etc/bash.bashrc	bash 配置文件	
系统环境	/etc/environment	环境变量
/etc/updatedb.conf	文件检索数据库配置信息	
/etc/issue	发行信息	
/etc/issue.net		
/etc/screenrc	屏幕设定	
网络	/etc/iftab	网卡 MAC地址绑定
/etc/hosts	主机列表	
/etc/hostname	主机名	
/etc/resolv.conf	域名解析服务器地址	
/etc/network/interfaces	网卡配置文件	

> ❶ 请使用“visudo”命令修改此文件，而不要直接编辑

用户配置文件

`/etc/` 目录下的文件，只有 root用户 才有权修改。应用软件的全局配置文件，普通用户也不应该修改，因为所有用户都要用到。

如果要通过配置软件，来适应特殊需求，您可以修改用户配置文件。

用户配置文件通常为全局配置文件的同名隐藏文件，放在 `$HOME` 目录下，例如：

全局配置文件	用户配置文件
/etc/inputrc	/home/user/.inputrc
/etc/vim/vimrc	/home/user/.vim/vimrc
/etc/bash.bashrc	/home/user/.bashrc ❶

> ❶ 也有少数例外，通常是系统程序

环境变量

环境变量是作用在整个系统中的变量。

很多软件工作的时候都要读取环境变量[21]的值来确定其工作方式。例如 cd 这个程序，如果不带任何选项执行，它会读取 `$HOME` 这个变量的值，然后进入到这个目录，也就是用户的 HOME 目录。使用 **echo** 显示一个环境变量：

```
echo $HOME
/home/user

echo HOME
HOME
```

> ❶ 依据惯例，变量名称通常使用大写字母；`$` 告诉 echo 这是一个变量，不要将字符串“HOME”直接输出到屏幕
>
> ❷ 输出字符串“HOME”到屏幕

```
WELCOME="Hello!"
echo $WELCOME
Hello!
env | grep WELCOME
export WELCOME
env | grep WELCOME
Hello!
```

> ❶ 定义一个变量
>
> ❷ 输出这个变量
>
> ❸ env 输出所有环境变量
>
> ❹ 将输出通过管道发送到 grep，检查其中是否有“WELCOME”这个值
>
> ❺ export 将变量 WELCOME 导出为环境变量

定义一个环境变量，可以将定义变量和导出变量一起进行

```
export WELCOME ="Hello!"
```

> ❶ 实际上不会有什么程序要用到 `$WELCOME` 这个变量，定义它只是为了演示

常用环境变量

变量名	说明
SHELL	默认 Shell
HISTSIZE	历史纪录
PATH	搜索路径
EDITOR	默认编辑器
TERM	默认终端
HOME	用户目录
LANG	系统语言
XMODIFIERS=@im	默认输入法

> [21] 必须由大量不同程序共享，独立于应用程序的优先选项，通常由环境变量指定，
>
> 如果通过配置文件给每个程序分别指定，既麻烦又容易出差错。
>
> 例如语言环境，如果所有能够使用多语言的程序都使用自己的配置文件确定其工作界面的语言，逐一去修改它们的配置文件将是一件苦恼的事情

第 16 章 软件管理

目录

软件管理方式

Linux 下安装软件，大致有两种方式：二进制和源代码。如何使用源代码安装软件，参阅第 18 章 编译工具链

二进制预编译软件包，可以快速安装部署你所需要的软件，但是你不能决定软件的特性。就像在麦当劳吃饭，只要几分钟，你就可以填饱肚子。但假设你不吃辣椒，或者你喜欢口感老一点的，你不能要求他们为你定做一份没有辣椒的汉堡，或者把鸡翅炸上半个小时；因为配方和工序都是固定的

使用源代码安装软件，虽然比较耗时，但可以满足你的特殊要求。就像在餐馆里吃饭，你可以要求厨师为你作不含辣椒的菜以适应你的口味(软件)，还可以要求厨师控制火候，作的老一点，以适应你的肠胃(硬件)

预编译软件包

虽然不能定做，但依然有许多人会去吃麦当劳，因为麦当劳的汉堡符合大多数人的口味；同样，预编译软件包也可以满足大多数用户的需求，如果没有特殊需求，很少有人会选择编译软件

最早的软件包管理系统不支持网络，类似于在 Windows 下安装软件，你必须获取一个安装包(比如通过网络下载)，然后在本地安装它。使用这种方式安装软件的包管理系统主要有：dpkg(debian系)、rpm(redhat系)

由于 Linux 系统中，软件包拆分的比较细[22]，安装一个软件可能要许多软件包。

而用户自己动手获取这些软件包，是一个十分繁琐的过程，因此产生了通过网络安装软件的方式：包管理系统自动分析你将安装的软件需要哪些软件包，通过网络下载、安装、配置它们。

debian系 发行版通过 apt 系统实现网络支持，配合本地的 dpkg 实现这一过程；redhat系 发行版则通过 yum 和 rpm 实现。而一些新派发行版，包管理系统兼具这两种能力，例如 archlinux 的 pacman；而 gentoo 的 emerge 虽然基于源代码编译安装，但也具备这种能力，参见“emerge”一节

以下是常见软件包管理系统的基本操作：

表 **16.1.** 包管理系统

<table>
<tr><th>任务</th><th>apt</th><th>zypp</th><th>yum</th><th>pacman</th><th></th></tr>
<tr><td></td><td>Debian, Ubuntu</td><td>openSUSE</td><td>Fedora, CentOS
(redhat系)</td><td>Archlinux</td><td>Ge</td></tr>
<tr><td>安装包</td><td>apt-get install pkg</td><td>zypper
install pkg</td><td>yum install pkg</td><td>pacman -S pkg</td><td>em</td></tr>
<tr><td>移除包</td><td>apt-get remove
pkg</td><td>zypper
remove pkg</td><td>yum erase pkg</td><td>pacman -R pkg</td><td>em</td></tr>
<tr><td>更新包列表</td><td>apt-get update</td><td>zypper
refresh</td><td>yum check-
update</td><td>pacman -Sy</td><td>em
| la
(添</td></tr>
<tr><td>更新系统</td><td>apt-get upgrade</td><td>zypper
update</td><td>yum update</td><td>pacman -Su</td><td>em
avu</td></tr>
<tr><td>列出源</td><td>cat
/etc/apt/sources.list</td><td>zypper
repos</td><td>yum repolist</td><td>cat
/etc/pacman.conf</td><td>lay</td></tr>
<tr><td>添加源</td><td>edit
/etc/apt/sources.list</td><td>zypper
addrepo 仓库地址 仓库名称</td><td>add 仓库 to
/etc/yum.repos.d/</td><td>edit
/etc/pacman.conf</td><td>lay</td></tr>
<tr><td>移除源</td><td>edit
/etc/apt/sources.list</td><td>zypper
removerepo
仓库名称</td><td>remove 仓库
from
/etc/yum.repos.d/</td><td>edit
/etc/pacman.conf</td><td>lay</td></tr>
<tr><td>搜索包</td><td>apt-cache search
pkg</td><td>zypper
search pkg</td><td>yum search pkg</td><td>pacman -Qs pkg</td><td>em
sea</td></tr>
<tr><td>列出已安装的包</td><td>dpkg -l</td><td>rpm -qa</td><td>rpm -qa</td><td>pacman -Qii</td><td>cat
/va
| m</td></tr>
</table>

- 把 SUSE系 和 redhat系 排在前面，不是因为它们的软件包管理系统比较先进，而是考虑

到它们的用户多

> [22] 因为不受版权的限制，凡是可以被其它软件利用的功能，都单独分离出来

第 17 章 核心工具集

目录

rm 目标目录|文件

rmdir 目标目录

mv 源文件 目标目录|文件

ln 源文件 链接

文件操作

split 源文件 [目标文件名前缀]

cat 文件名

sort [-o 输出文件] [-t 分隔字符] [+起始字段 -结束字段] [文件]

more

diff 文件1 文件2

patch

cksum [文件名]

md5sum [文件名]

压缩解压

tar [-]c|x|u|r|t[z|j][v] -f 归档文件 [待打包文件]

7z|7za 子命令 [选项] 压缩包 [文件]

搜索

whereis 程序名称

locate 文件名称

find [路径] 表达式

grep 字符串|正则表达式 文件名|标准输入

权限管理

细节

chmod 权限表达式 文件|目录

chown 归属用户[:归属群组] 文件|目录

chgrp 归属群组 文件|目录

SUID、SGID、Sticky bit

lsattr [路径]

chattr +|-|=属性 路径

用户管理

细节

su [用户名]

sudo 命令

passwd [用户名]

gpasswd 群组名

chsh -s Shell [用户名]

usermod 用户名

useradd 用户名

userdel 用户名

id [用户名]

进程管理

细节

ps

pstree [进程编号 | 用户]

pgrep 进程名

kill [信号代码] 进程编号

pkill [信号代码] 进程名称

xkill

top

renice 优先级 进程

nohup 命令 [选项]

命令 &

命令1 ; 命令2 ; ......

命令1 && 命令2 && ......

Ctrl+z

jobs

bg [任务编号]

fg [任务编号]

磁盘和内存管理

细节

mount 设备文件 [挂载路径]

umount 设备文件 | 挂载路径

df

free

sync

fdisk 磁盘设备文件

cfdisk

mkfs.文件系统类型 分区设备文件

hdparm 磁盘设备文件

硬件管理

lspci

lsusb

lsmod

modprobe 模块名称

网络管理

其它

echo 字符串

clear

alias 输入内容=实际内容

export 变量名

shutdown

halt

reboot

init 数字

chroot 路径

细节

Linux 是大小写敏感的系统，所有的命令、路径、选项、参数、变量……都区分大小写

使用 `TAB` 键补全命令，无论任何时候，多按几次 TAB 总会有所帮助

Shell 的功能键能够协助您更高效的编辑命令，请熟悉其键绑定，尽量使用它

命令由 `命令名` 、 `分隔符` 、 `选项` 、 `操作对象` 构成

命令名

标识命令的功能，例如**cp**(copy)、**mv**(move)、**rm**(remove)……

有些命令包含一些子命令，您可以认为它的命令名由两个单词构成，例如“apt”软件包管理系统：

```
apt-get install            安装一个软件
apt-get remove             删除一个软件
```

分隔符

通常为空格，多个连续的空格视为一个空格，下面两个命令相同：

```
cp a b
cp    a     b
```

有一些特殊符号也属于分隔符，例如管道 `|` 、重定向 `>` 、 `>>` 、 `<` 、后台运行 `&` 、序列执行 `&&` 、 `;` 。使用这些符号时，您不需要再使用空格作为分隔符

```
ls -al|less     #两个命令效果相同
ls -al | less     #更容易阅读
```

选项

精细调节命令的行为，以 `-` 引导，通常为选项的首字母。许多软件都可以使用 **-h** 选项来阅读使用说明，例如：

```
apt-get -h
apt-get --help
apt-get -help
apt-get -h -e -l -p
```

> ❶ 也可以使用选项的全名，一般以 `--` 引导
>
> ❷ 多数命令中，使用 `-` 引导多个字符，将会被视为多个选项
>
> ❸ 与上面的命令效果一样

少数命令的选项，不需要以 `-` 引导，或者使用 `-` 引导选项全名，例如：

```
ps aux
/etc/init.d/gdm start
mplayer -loop xxx
```

递归。表示在子层次中重复相同操作。例如递归复制某目录，不但复制当前目录及其下的所有文件；而且对当前目录的子目录，也进行递归复制的操作。

格式约定[23]

使用 `[]` 表示可选项，实际输入为方括号中的内容，例如

```
ls [-l]
```

- 实际输入为，`ls -l`

使用 `|` 表示"或"，以 | 分隔的项目不能同时使用，例如

```
tar [-z|j c|x vf] 归档文件 [源文件]
```

- 选项通常紧跟命令名，除非必要，在命令格式中，我们通常省略它们

> [23] 与man中的约定相同

系统信息

uptime

联机信息-时间，显示如下

```
11:27pm    up 9 days,  7:12,      3 user,  load average:   0.07,   0.12, 0.14
```

> ❶ 当前系统时间
>
> ❷ 系统运行时间
>
> ❸ 当前在线用户数
>
> ❹ 系统负荷:1分钟前、5分钟前、15分钟前

w

联机信息-已登录用户，显示如下

```
01:04:10 up  1:34,  2 users,  load average: 0.25, 0.16, 0.11
USER     TTY   FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
user     tty1     192.168.0.1       23:30    1:33   0.14s  0.12s -bash
```

> ❶ uptime 信息
>
> ❷ 用户名
>
> ❸ 登录方式
>
> ❹ 来源地址
>
> ❺ 登录时间
>
> ❻ 发呆时间
>
> ❼ 资源占用
>
> ❽ 当前任务

who

联机信息

-r	运行级别

whoami

显示当前用户名

last

最近用户登录信息

-数字	使用数字作为选项，控制显示条目

uname

系统信息

-s	内核名称（默认选项）
-a	全部
-p	CPU 信息
-n	主机名
-r	内核发行信息（版本号）
-v	内核版本信息

date

显示、设定系统时间

-u	显示格林尼洛时间（UTC）	
MMDDhhmm[[CC]YY][.ss]	设定时间，需要管理员权限。	date 12292359
MM	月份	
DD	天数	
hh	小时	
mm	分钟	
CC	年份前两位	
YY	年份后两位	
ss	秒钟	
秒钟、年份为可选	date 122923592006.59	
+[%X]	设定显示格式，默认输出格式：	date +%Y年%m月%d日%A%H:%M:%S%Z
格式控制	%n	换行
%t	制表符	
小时	%H	00~23

%I	01~12	
%k	0~23	
%l	1~12	
%p	AM|PM	
分、秒	%M	分钟(00~59)
%S	秒(00..61)	
%T	hh:mm:ss	
%r	hh:mm:ss AM|PM	
%s	从 1970年1月1日00:00:00 UTC 到目前为止的秒数	
%X	%H:%M:%S	
%Z	时区	
星期	%a	Sun~Sat
%A	Sunday~Saturday	
%w	一周中的第几天 (0~6)	
年份	%Y	0000~9999
%y	00~99	
月份	%m	01~12
%b	Jan~Dec	
%h		
%B	January~December	
日期	%c	完整日期时间
%d	01~31	
%j	001~366	
%x	本地格式日期	
%D	mm/dd/yy	
一年中的第几周	%U	以 Sunday 为一周的第一天
%W	以 Monday 为一周的第一天	

cal

显示日历

文件管理

细节

`/` 目录为文件系统根目录，所有目录都是它的子目录

绝对路径以 `/` 起始，相对路径以当前所在目录起始

目录是一种特殊类型的文件，如果没有特别指明， 文件 包括文件和目录

`..` 表示上一级目录， `.` 表示当前目录，它们是两个特殊目录

链接. 为当前文件建立在其它路径中的访问方法。例如将系统中其它位置的可执行文件，链接到 `/usr/local/bin` 目录下，使用命令调用。

ls [路径]

显示当前目录文件列表

--color	不同属性以不同颜色显示（默认选项）
-a	全部显示
-i	显示 inode值
-l	详细信息
-F	显示文件类型后缀 目录 `/` 链接 `@` 可执行文件 `*` 端口文件 `=` 管道文件 `|`
-A	显示隐藏文件
-R	递归显示子目录文件列表
-S	按文件大小排序
-t	按修改时间排序
-u	按访问时间排序
-d	只显示目录，不递归显示目录下的文件

cd [目录路径] | [特殊路径]

切换目录,目录可以使用绝对路径或者相对路径

~	$HOME 目录（默认值）
-	上一次目录
..	上一级目录
.	当前目录

- 您可以通过修改 `/etc/environment` 文件，来定义 `$CDPATH` 变量，设定“**cd**”命令的搜索路径。

pwd

显示当前路径

file 文件名

显示文件类型

-i	显示 mime类型

du [路径]

计算文件或目录空间占用

-h	人性化显示。自动以 G、M、K 为单位显示占用空间大小
--max-depth=数字	显示目录深度
-l	重复计算硬链接文件大小
-L	计算符号链接文件大小
-a	显示当前目录子目录中的文件
-c	显示文件数

less 文件名

浏览文件，使用 VI 和 Emacs 两种风格的键绑定。

touch 目标文件

触碰，在不修改文件的前提下，更改其时间属性。通常用来创建一个空文件

mkdir 文件夹

创建文件夹

-p 多级目录	按路径创建多级目录
-m 数字权限值	设定权限

cp 源文件 目标目录|文件

将源文件复制为目录文件，或者将源文件复制到目标目录。多个源文件使用空格分隔

cp 源目录 目标目录

将源目录复制到目标目录中，如果复制多个源目录，需要使用 **-R** 选项

-a	相当于 -dpr 选项
-d	保留链接
-f	强制复制，覆盖目标文件
-i	覆盖时询问用户
-p	保留修改时间和访问权限
-r -R	递归复制（目录=>目录）
-l	创建链接
-v	显示过程

rm 目标目录|文件

删除

-r -R	递归删除（用于删除目录）
-f	强制删除（无需确认，直接删除。慎用！）
-i	交互式删除（询问用户）

rmdir 目标目录

删除目录时，建议您使用“**rm -r**”命令

mv 源文件 目标目录|文件

相当于 cp 后删除源文件，也可以作为“重命名”使用。

-f	强制，覆盖目标文件
-i	覆盖时询问用户
-v	显示过程

ln 源文件 链接

链接

-s	符号链接
-f	强制链接，覆盖目标文件
-i	覆盖时询问用户

文件操作

split 源文件 [目标文件名前缀]

将源文件按一定规则分割成若干个目标文件。默认文件名前缀为 `x`

-行数	按行数分割文件
-l 行数	同上
-b 字节	按大小分割文件。可以使用 b、k、m 作单位，不指定单位的情况下，默认单位为 b
-C 字节	按大小分割文件，并尽量保持每行的完整

cat 文件名

输出文件内容。用空格分隔多个文件名

-n	在输出中添加行号
-b	在输出中添加行号，空行不编号
-s	将两行或以上的空行，合并为一个空行

- 可以将多个文件内容连接到一起输出。使用重定向合并为一个文件 **cat xaa xab xac > file.split**

sort [-o 输出文件] [-t 分隔字符] [+起始字段 -结束字段] [文件]

对文本内容排序

-m	合并文件
-c	检查文件是否已按规则排序
-b	忽略行首空格字符
-u	忽略内容重复行
-f	忽略大小写
-l	忽略非打印字符
-M	作为月份比较
-d	按字典顺序排序，按照字母、数字、空格、制表符排序
-r	逆序输出

more

查看文件内容，建议使用 less

diff 文件1 文件2

比较文件

无选项	混合 ed 命令格式
-u [数字]	统一格式，数字为显示上下文行数
--unified[=数字]	
-c [数字]	上下文格式，数字为显示上下文行数
--context[=数字]	
-e	ed 命令格式
--ed	
-f	RCS 命令格式

patch

使用 **diff -uN[r]** 旧文件 新文件 **>** 补丁文件 命令创建补丁文件。

在待补丁文件的目录下使用 **patch -p[数字] <**补丁文件 命令打补丁

-p [数字] 表示忽略补丁文件中记录的目录，数字为忽略的层数。

patch -R 已打补丁文件 补丁文件 将已打补丁文件恢复到原来的状态

cksum [文件名]

计算文件的 CRC值。不指定文件名则从标准输入设备读入数据

md5sum [文件名]

计算文件的 md5值。不指定文件名则从标准输入设备读入数据

-t	以文本模式读取
-b	以二进制模式读取
-c md5纪录	校验 md5纪录 中的文件(使用 md5sum 配合重定向生成纪录文件)

压缩解压

tar [-]c|x|u|r|t[z|j][v] -f 归档文件 [待打包文件]

将多个文件打包为一个归档文件，可以在打包的同时进行压缩。支持的格式为 tar（归档）、gz（压缩）、bz2（压缩率更高，比较耗时）

操作选项	-c	创建
-x	解包	
-u	更新	
-r	添加	
-t	查看	
-d	比较压缩包内文件和文件	
-A	将 tar 文件添加到归档文件中	
格式选项	-z	使用 gz 压缩格式
-j	使用 bz2 压缩格式	
其它	-v	显示过程
-f 文件名	归档文件的文件名，使用 `-` 代表标准输入/输出	
-C 解压路径	将压缩包中的文件解压到指定目录	
--exclude=文件	排除文件	
-P	使用绝对路径压缩时，保留根目录“/”	
-W	校验	
-p	还原文件权限	
-w	询问用户	
--totals	统计	
-T 文件列表	处理文件列表中的文件	
-X 文件列表	排除文件列表中的文件	

7z|7za 子命令 [选项] 压缩包 [文件]

子命令	a	添加
d	删除	
e	解压	
x	带路径解压	
l	列表查看	
t	测试	
u	更新	
选项	-m 压缩方式	
-m0=压缩算法	默认使用 lzma	
-mx=数字	1~9 压缩级别	
-mfb=64	number of fast bytes for LZMA = 64	
-md=字典大小	设置字典大小，例如 -md=32m	
-ms=on|off	是否固实压缩	
-o输出目录	设置输出目录	
-p密码	使用密码	
-r数字	递归，使用数字定义递归子目录的深度	
-sfx[模块名称]	使用自解压模块	
-si	从标准输入设备读入数据	
-so	将数据写入标准输出设备	
-y	所有询问均回答 Yes	
-w路径	设置工作目录	

搜索

whereis 程序名称

查找软件的安装路径

-b	只查找二进制文件
-m	只查找帮助文件
-s	只查找源代码
-u	排除指定类型文件
-f	只显示文件名
-B 目录	在指定目录下查找二进制文件
-M 目录	在指定目录下查找帮助文件
-S 目录	在指定目录下查找源代码

locate 文件名称

在文件索引数据库中搜索文件

-d 数据库路径	搜索指定数据库

- `updatedb` 更新文件索引数据库

find [路径] 表达式

查找文件

-name 表达式	根据文件名查找文件
-iname 表达式	根据文件名查找文件，忽略大小写
-path 表达式	根据路径查找文件
-ipath 表达式	根据路径查找文件，忽略大小写
-amin 分钟	过去N分钟内访问过的文件
-atime 天数	过去N天内访问过的文件
-cmin 分钟	过去N分钟内修改过的文件
-ctime 天数	过去N天内修改过的文件
-anewer 参照文件	比参照文件更晚被读取过的文件
-cnewer 参照文件	比参照文件更晚被修改过的文件
-size 大小	根据文件大小查找文件，单位 b c w k M G
-type 文件类型	根据文件类型查找文件。b 块设备 c 字符设备 d 目录 p 管道文件 f 普通文件 l 链接 s 端口文件
-user 用户名	按归属用户查找文件
-uid 用户编号	按用户编号查找文件
-group 群组名	按归属群组查找文件
-gid 群组编号	按群组编号查找文件
-empty	查找空文件

grep 字符串|正则表达式 文件名|标准输入

在文件中搜索内容

权限管理

细节

一个文件主要包含下列属性， `ls -l`

```
- rwxrwxrwx  user  group  date   filename
  111 101 101
```

> ❶ 归属用户的权限
>
> ❷ 归属群组的权限
>
> ❸ 其它用户的权限
>
> ❹ 归属用户
>
> ❺ 归属群组
>
> ❻ 日期信息
>
> ❼ 文件名称

- 对于文件夹，必须拥有它的可执行权限，才能够使用 **cd** 命令进入该文件夹；拥有可读权限，才能够使用 **ls** 命令查看该文件夹的文件列表。
- root用户 拥有最高权限

可以使用 3位的二进制数字 来描述一组权限，某一权限对应的数字为 1,则表示具有该种权限，为 0,则不具有该种权限。

使用二进制数字来描述一组权限，虽然非常直观，但是 3组 权限需要用 9位 数来表示，不够方便。因此我们将三组权限使用 3位8进制数字 来表示。

每种权限对应的数字:

权限	r	w	x
二进制	100	010	001
八进制	4	2	1

将这 3位8进制数字 相加的结果，就可以表示该组权限的具体内容，例如：

```
7=4+2+1=rwx
5=4+1=rx
755=4+2+1,4+1,4+1=rwx,r-x,r-x
```

还可以使用 a 、 u 、 g 、 o 表示归属关系，使用 = 、 + 、 - 表示权限变化，使用 r 、 w 、 x 表示权限内容

```
a 所有用户   u 归属用户   g 归属群组   o 其它用户
= 具有权限   + 增加权限   - 去除权限
r 可读权限   w 可写权限   x 可执行权限
```

例如：

```
a+x 给所有用户增加可执行权限
go-wx 将归属群组和其它用户的可写、可执行权限去掉
u=rwx 归属用户具有可读、可写、可执行权限
```

chmod 权限表达式 文件|目录

更改文件的权限。权限的表达式可以使用 3位8进制数字 表示，或者使用 augo +-= rxw-s 来表示

-R	递归
-v	显示过程
-c	类似“-v”，仅显示更改部分
--reference=参照文件或目录	以指定文件为参照更改权限

示例：

```
chmod -R a+x path
chmod -Rv 755 path
```

chown 归属用户[:归属群组] 文件|目录

更改文件的归属用户。可以使用用户名或者用户编号

-R	递归
-v	显示过程
-c	类似“-v”，仅显示更改部分
--reference=参照文件或目录	以指定文件为参照更改权限

示例：

```
chown user:admin path
chown -R user.admin path
chown user path
```

chgrp 归属群组 文件|目录

更改文件的归属群组。可以使用群组名或者群组编号,选项同上

SUID、SGID、Sticky bit

某些情况下，需要以可执行文件归属用户的身份执行该文件，可以为该文件设置 SUID。同样，设置 SGID 能够以该文件归属群组的身份执行它。

例如：用户自行设定密码。出于安全方面的考虑， `/etc/shadow` 只能由 root用户 直接修改。

```
-rw------- root root /etc/shadow
```

这个时候，可以为程序 /usr/bin/passwd 设置 SUID，当普通用户执行“**passwd**”命令时，便能够以该程序归属用户 root 的身份修改 /etc/shadow 文件。而“passwd”程序自身带有身份验证机制，不能通过验证时拒绝执行，从而保证了安全。

```
ls -l /usr/bin/passwd
-r-s--x--x root root /usr/bin/passwd
```

我们发现，归属用户的可执行权限位使用 *s* ，表示 SUID。同样，归属群组的可执行权限位使用 *s* ，表示 SGID。任何用户或群组都拥有 “其它用户” 的权限，所以不需要以 其它用户 身份执行文件，其它用户的可执行权限位便不会出现 *s* 。该权限位可能出现的属性为 *t* ，也就是粘着位 Sticky bit。

```
ls -ld /tmp
drwxrwxrwt root root /tmp
```

粘着位表示任何用户都可能具有写权限，但只有该归属用户或 root用户 才能够删除

SUID、SGID、Sticky bit 也可以像权限一样，使用一个八进制数表示，如下：

4	SUID
2	SGID
1	Sticky bit

通过在“**chmod**”命令中使用 4个八进制数 的表达式，如 4755 ，用第一位表示 SUID、SGID 或 Sticky bit，便能够为文件设置这些特殊权限。示例：

```
chmod -R 4755 path
```

lsattr [路径]

查看文件的特殊属性

-a	全部显示
-d	只显示目录
-R	递归

特殊属性包括：

a	仅供附加用途
b	不更新最后存取时间
c	压缩后存放
d	排除在转储操作之外
i	不得任意更动文件或目录
s	保密性删除文件或目录
S	即时更新文件或目录
u	预防意外删除

chattr +|-|=属性 路径

更改文件特殊属性

-R	递归
-V	显示过程

用户管理

细节

root 用户为根用户,也就是系统管理员,拥有全部权限

一个用户只能拥有一个 群组编号 ，但是还可以归属于其它附加群组

用户管理的重要配置文件：

/etc/passwd	用户名 密码位 用户编号 归属群组编号 姓名 $HOME目录 登录Shell
/etc/shadow	用户名 已加密密码 密码改动信息 密码策略
/etc/group	群组名 密码位 群组编号 组内用户
/etc/gshadow	群组密码相关文件
/etc/sudoers	用户名 权限定义 权限

- 请使用“**visudo**”命令修改 `/etc/sudoers` ，而不要直接编辑
- 可以使用 **pwconv** 命令创建影子密码，将 `/etc/passwd` 文件中的密码转换到 `/etc/shadow` 文件

su [用户名]

切换到其它用户，默认切换到 root用户 。提示密码为目标用户密码

-f	快速切换，忽略配置文件
- | -l	重新登录
-m |-p	不更改环境变量
-c 命令	切换后执行命令，并退出切换

sudo 命令

以其它用户的身份执行命令，默认以 root 的身份执行。提示密码为当前用户密码

-s	切换为 root shell
-i	切换为 root shell，并初始化
-u 用户名|用户编号	执行命令的身份
-l	显示自己的权限

passwd [用户名]

设定用户密码

-d	清除密码
-l	锁定账户
-e	使密码过期，在下次登录时更改密码
-S	显示密码认证信息
-x 天数	密码过期，最大使用时间
-n 天数	冻结密码，最小使用时间
-s	更改 登录Shell
-f	更改用户信息

示例：

```
$passwd
Changing password for user
(current) UNIX password:      #原密码
Enter new UNIX password:      #新密码
Retype new UNIX password:     #确认新密码
```

gpasswd 群组名

更改群组

-a 用户名	将用户加入群组
-d 用户名	将用户从群组中删除
-r	删除密码
-A 用户名	将用户设置为群组管理员(群组管理员或 root 才可以使用 gpasswd 命令)
-M 用户1,用户2……	设置群组成员

chsh -s Shell [用户名]

更改登录 Shell

usermod 用户名

修改用户账号

-d 目录	设定 $HOME 目录
-m	设定 $HOME 目录时自动建立该目录
-s Shell	修改用户 登录Shell
-l 新登录名	修改为新登录名
-u 用户编号	修改用户编号
g 群组名	修改用户归属群组
-G 群组名	修改用户归属辅组
-L	锁定帐户
-U	解除锁定
-e 过期时间	设定用户账号过期时间
-f 缓冲天数	设定密码过期后多长时间关闭账号
-c 字符串	修改用户备注

useradd 用户名

新建用户

-d 目录	设定 $HOME 目录
-m	自动建立 $HOME 目录
-M	不自动建立 $HOME 目录
-s Shell	设定用户 登录Shell
-u 用户编号	设定用户编号
-g 群组名	设定用户归属群组
-G 群组名	设定用户归属附加群组
-n	不建立以用户名为名称的群组
-e 过期时间	设定用户账号过期时间
-f 缓冲天数	设定密码过期后多长时间关闭账号
-c 字符串	设定用户备注
-D [表达式]	更改预设值 （预设值保存于 /etc/default/useradd 文件中）

- 新建用户规则保存于 `/etc/login.defs` 文件中
- 新建用户默认文件保存于 `/etc/skel/` 目录中。新建用户时，系统自动拷贝此目录下的文件至新建用户的 `$HOME` 目录

userdel 用户名

删除用户

-r	删除用户相关文件和目录

id [用户名]

显示用户 用户编号 群组编号 归属附加群组

进程管理

细节

进程一般分为批处理进程、交互进程和守护进程三类。

守护进程总是活跃，在系统启动时通过脚本自动启动，或由 root 启动，通常在后台运行。

一个进程可以拥有子进程。当父进程终止时，它的子进程也随之终止；而子进程终止时，父进程通常可以继续运行。

init 进程为根进程，所有进程都是它的子进程

ps

显示进程信息，选项可省略“`-`”

aux	以 BSD风格 显示进程 常用
-efH	以 SystemV风格 显示进程
-e | -A	显示所有进程
a	显示终端上所有用户的进程
x	显示无终端进程
u	显示详细信息
f	树状显示
w	完整显示信息
l	显示长列表

输出字段

USER	进程所有者
PID	进程编号
PPID	父进程编号
%CPU	CPU 占用率
%MEM	内存占用率
NI	进程优先级。数值越大，占用 CPU 时间越少
VSZ	进程虚拟大小
RSS	页面文件占用
TTY	终端编号
STAT	进程状态
D	不可中断
R	正在运行，或在队列中的进程
S	处于休眠状态
T	停止或被追踪
Z	僵尸进程
X	死掉的进程
<	高优先级
N	低优先级
L	有些页被锁进内存
s	包含子进程
+	位于后台的进程组
l	多线程，克隆线程

pstree [进程编号 | 用户]

树状显示进程信息。可选择显示某用户的进程或从某进程编号开始的进程

-a	显示完整命令及选项
-c	完全显示重复进程
-p	显示进程编号，隐含-c
-n	按进程编号排列进程
-u	显示进程所有者
-h	
-H 进程编号	高亮显示进程编号指定的进程及其祖先

pgrep 进程名

显示进程编号

-l	显示进程名和进程编号
-o	进程起始编号
-n	进程终止编号

kill [信号代码] 进程编号

根据进程编号向进程发送信号，常用来结束进程，默认信号为 -9

-l [信号代码]	显示、翻译信号代码
-9 | -KILL	发送 kill 信号,退出
-6 | -ABRT	发送 abort 信号,退出
-15 | -TERM	发送 Termination 信号
-1 | -HUP	挂起
-2 | -INT	从键盘中断，相当于 Ctrl+c
-3 | -QUIT	从键盘退出，相当于 Ctrl+d
-4 | -ILL	非法指令
-11 | -SEGV	内存错误
-13 | -PIPE	破坏管道
-14 | -ALRM	
-STOP	停止进程，但不结束
-CONT	继续运行已停止的进程
-9 | -1	结束当前用户的所有进程

pkill [信号代码] 进程名称

结束进程族。如果结束单个进程，请用 kill

xkill

在图形界面中点杀进程。执行此命令后，鼠标指针变为骷髅图案（一定看过《加勒比海盗》吧）。在窗口中点击左键杀死进程，右键取消

top

动态、交互式进程管理器

启动选项	-b	
-c	显示进程启动状态，包括选项、参数、操作对象等；而不只是进程名	
-d 秒	刷新频率。 -d 5，表示5秒刷新一次	
-n 次	刷新次数，然后退出。 -n 5，表示刷新5次后退出	
-i	禁止显示空闲进程或僵尸进程	
-p 进程编号	仅监视指定进程的编号	
-s	安全模式运行，禁用一些交互指令	
-S	累积模式，输出每个进程的总的 CPU 时间，包括已死的子进程	
交互命令	space	立即刷新
k	交互式杀死进程，提示输入进程编号（默认发送信号15）	
r	设定 renice，提示输入进程编号和 renice值	
s	改变两次刷新时间间隔，以秒为单位	
n	设定显示进程数， 0 为不作限制	
i	隐藏空闲进程和僵尸进程	
S	切换到累积时间模式	
l	开关，在顶部显示 uptime 信息	
t	开关，在顶部显示 进程和 CPU 状态	
m	开关，在顶部显示 free 信息	
c	显示方式切换： 进程名/进程启动状态	
A	按进程启动顺序进行排序。由新到旧	
M	按内存占用排序。由大到小	
N	以进程编号排序。由大到小	
P	按 CPU 占用排序。由大到小	
T	按时间／累积时间排序	
f|F	设定显示字段。设定完成后空格退出	
o|O	设定显示字段的排序。大写向前移动，小写向后移动，空格退出	
h|?	显示有关安全模式和累积模式的帮助信息	
W	把当前的配置写到 ~/.toprc 中	

renice 优先级 进程

重新设定进程优先级（通常无此必要）

<table>
<tr><th></th><th></th><th></th></tr>
<tr><td>优先级表达式</td><td>+|-|= nice值</td><td></td></tr>
<tr><td>nice 取值范围</td><td>-20~19</td><td></td></tr>
<tr><td>进程表达式</td><td>-p 进程编号</td><td>通过进程编号进行设定</td></tr>
<tr><td>-g 进程群组编号</td><td>通过进程群组编号</td><td></td></tr>
<tr><td>-u 用户编号</td><td></td><td></td></tr>
</table>

nohup 命令 [选项]

将任务提交到后台，输出附加到 `~/nohup.out` 文件。即便用户退出登录，提交的命令仍继续执行。

命令 &

背景执行此命令，如果用户退出登录，则命令停止执行

命令1 ; 命令2 ; ......

命令队列，从左向右，依次执行以 `;` 分隔的命令

命令1 && 命令2 && ......

命令队列，从左向右，依次执行以 `&&` 分隔的命令。前一个命令执行成功，后一个命令才能执行

Ctrl+z

挂起当前任务

jobs

显示背景任务

-l	显示完整信息

bg [任务编号]

将挂起的任务背景执行

fg [任务编号]

将背景任务调到前台执行

磁盘和内存管理

细节

Linux 中，设备用 `/dev/` 目录下的文件表示。例如

```
/dev/hda1 第一块硬盘的第一主分区
/dev/hdb5 第二块硬盘的第一逻辑分区
/dev/sda4 第一块 SATA 硬盘的第四主分区，或者扩展分区
/dev/null 黑洞设备
```

关于磁盘设备，详见“分区概念”一节

mount 设备文件 [挂载路径]

挂载文件系统

-t	指定文件系统的类型	通常不必指定，mount 自动检测
常见类型	reiserfs	ReiserFS 3.6版
xfs	SGI 技术	
jfs	IBM 技术	
ext3	Linux 传统文件系统	
vfat	fat fat32	
ext2	Linux 传统文件系统，不带日志	
ntfs	WINNT	
iso9660	光盘	
smbfs	Windows 文件共享	
-o [选项1] [选项2]		
选项	loop	环设备。光盘、ISO 映像文件等，通常用于挂载映像文件（而不是设备文件）

bind	绑定。将一个目录（而不是设备文件）挂载到另一个目录
ro | rw	只读 readonly；可读写 read-write
sync | async	同步模式|异步模式。决定修改是否立即写入文件系统
atime | noattime	读取时是否修改访问时间。对于写入敏感设备，例如闪存、软盘，建议使用 noatime
auto | noauto	自动挂载模式
exec | noexec	是否允许执行
defaults	使用预设的选项 rw, suid, dev, exec, auto, nouser, async
iocharset=UTF-8	指定字符集，可简写为 utf8
codepage=936	指定代码页，可简写为 cp936 西文系统代码页为 437
umask=权限掩码❶	设定权限掩码
uid=用户编号	设定归属用户
gid=群组编号	设定归属群组
remount	以不同选项重新挂载
-L 卷标	挂载带有特殊卷标的分区

> ❶ 权限掩码=777-目标权限（三位） | 7777-目标权限（四位）假如权限掩码为 022 ，则目标权限为 755 rwxr-xr-x可以使用 **umask** 命令设置权限掩码
>
> 提示：挂载 NTFS 分区时请使用 **utf8** 选项；挂载 FAT 分区时请使用 **iocharset=utf8,codepage=936** 选项，以避免乱码

mount -a

挂载 `/etc/fstab` 文件中定义的所有设备。示例：

```
sudo mount -t iso9660 -o loop /dev/cdrom0 /media/cdrom
sudo mount -t vfat -o remount iocharset=utf8,codepage=936  /dev/hda5 /media/hda5
```

umount 设备文件 | 挂载路径

卸载已挂载文件系统

df

查看已挂载文件系统的磁盘空间占用

-a	显示所有文件系统的磁盘使用情况，包括0块（block）的文件系统，如 `/proc 文件系统`
-T	显示文件系统类型
-k	以 k 字节为单位显示
-i	显示 i节点 信息，而不是磁盘块
-t 文件系统类型	显示指定类型的文件系统的磁盘空间使用情况
-x 文件系统类型	列出排除指定类型文件系统的磁盘空间使用情况（与 t 选项相反）
-l	只显示本地文件系统

free

查看内存、缓冲区、交换空间的占用

-b	以字节为单位显示数值
-k	以千字节为单位显示数值
-m	以兆字节为单位显示数值
-g	以吉字节为单位显示数值
-l	显示内存占用峰值
-o	不显示缓冲区占用
-t	统计结果
-s 秒	刷新频率

sync

同步文件系统。将缓冲区中的数据写入文件系统

fdisk 磁盘设备文件

分区表修改工具(磁盘设备文件应为整块磁盘，而不是磁盘中的分区。例如 `/dev/sda` ，而不是 `/dev/sda1`)

m	使用帮助
l	查看已知文件系统类型
p	显示分区信息
n	新建分区 （p:主分区 l:扩展分区 参见“分区概念”一节 ）
d	删除分区
t	改变分区类型
w	将改动写入分区表
q	放弃改动并退出

fdisk -l 查看所有磁盘分区信息

cfdisk

更加友善的分区表修改工具

mkfs.文件系统类型 分区设备文件

将分区格式化为文件系统。示例：

```
sudo mkfs.reiserfs /dev/hda1
sudo mkfs.xfs /dev/sda1
```

hdparm 磁盘设备文件

设置硬盘参数

-d 0|1	DMA 模式开关
-a 0|1	预读模式开关
-t	性能测试
-T	缓存性能测试
-c 0|1|3	32位传输模式开关
-g	显示柱面，扇区等信息
-i|-I	显示磁盘信息

硬件管理

lspci

查看 PCI 总线连接的设备(实际上也可以查看 ISA、AGP、USB 等总线信息)

—v	显示详细信息
—vv	
—vvv	
—k	同时显示使用的驱动和加载的模块
—n	显示设备 ID号
—b	显示 PCI、ISA 扩展槽地址和中断
—t	显示总线树结构
-F 文件	从指定文件读信息
—m	以便于机器处理的格式输出所有硬件信息
-mm	

lsusb

查看 USB 接口连接的设备

lsmod

查看已加载模块。`/lib/modules/`**uname -r** 目录下为所有可用模块

modprobe 模块名称

启用模块

-a	加载所有匹配模块
-c	显示当前使用的配置
-d	显示调试信息
-k	将指定模块设置为"自动清除"模式
-l	显示所有匹配模块
-n	模拟执行
-q	不显示错误信息
-r	删除使用命令加载的模块;对非命令加载的模块设置"自动清除"模式
-s	将结果记录到系统记录中
-t	指定模块类型
-v	显示详细信息
-C	指定配置文件.默认使用 `/etc/modules.conf` 文件为配置文件

网络管理

其它

echo 字符串

回显。较复杂的字符串，可以使用 `"` 括起来。

选项	-n	输出内容不换行
-E	不解析脱字符	
-e	解析脱字符	
控制字符	\	反斜线
\a	警告	
\b	退格	
\n	换行	
\r	回车	
\t	水平制表符	

clear

清空屏幕

alias 输入内容=实际内容

别名，为命令指定一个别名，以简化输入。例如：

```
alias ls='ls -AF --color=auto'
```

- 可以将您的定义保存在 `~/.bashrc` 文件中

export 变量名

将变量导出为环境变量，常写变量赋值一同使用，例如：

```
export PATH="$PATH:xxx"
```

shutdown

关闭计算机，向根进程 init 发送信号，更改 runlevel 为 0 (halt)

-h	关闭电源
-r	重启
-n	强行关机，不向 init进程 发送信号
-k	模拟关机，向登录者发送关机警告
-t 秒	N秒后关机
time 时间	定时关机
-c [说明信息]	取消关机
-f	重启时忽略检测文件系统
-F	重启时强制检测文件系统

halt

关闭计算机。 调用 **shutdown -h** ，结束系统进程，同步文件系统，停止内核。

-n	不同步文件系统		
-w	模拟关机，写 `/var/log/wtmp` 纪录	-f	不调用 shutdown ,强行关机
-p	默认选项，关机时调用 poweroff		
-i	关机前断开网络		

reboot

重新启动计算机。选项与 halt 相似

init 数字

更改运行级别

0	关机
1	单用户模式
6	重启

chroot 路径

更改根目录，重新定义会话的运行环境。（通常用在多系统环境下使用）

第 18 章 编译工具链

目录

标准编译安装

为什么要编译安装

包管理系统是绝大多数发行版的必备组件，也是一个发行版区别于其它发行版的主要特征。但是有些软件，并不能通过包管理系统安装，这就需要下载源码编译安装。

一个软件可能有许多功能，但是发行版中提供的安装包，通常只具有一些常见的功能。如果提供所有功能，那么无疑会占用更多的资源，而这些功能，大多数用户不会用到；[24]而你会用到的功能，可能安装包中刚好没有。编译安装可以灵活地定制软件，选择自己需要的，取消自己不需要的。

编译安装还可以针对特定的硬件进行优化，以获得更好的性能表现。[25]

编译环境

编译环境包括多个工具，它们环环相扣，称作编译工具链。主要包括以下工具：

工具	简介
binutils	连接器、汇编器和其他用于目标文件和档案的工具
gcc	编译器，将源代码转换为机器代码
glibc	C库，提供标准例程(C函数)

还有一些工具，能够调用工具链，实现自动化编译：

autoconf	自动生成 `Makefile` 文件
automake	
make	按照 `Makefile` 文件中的规则编译程序

在后面的部分将分别介绍这些工具

标准编译安装

首先，下载源代码，通常是压缩包，如：`xxx.tar.gz` 或者 `xxx.tar.bz2` ，解包：

压缩包格式	命令
.tar.gz	tar zxvf xxx.tgz
.tgz	
.tar.bz2	tar jxvf xxx.tar.bz2

通常解包后会在当前位置得到一个 `xxx/` 目录，进入这个目录

```
cd xxx/
```

使用下列命令编译安装：

```
./configure --prefix=/opt/xxx
make
sudo make install
make clean
```

> ❶ 配置软件特性，检查编译环境，生成 `Makefile文件`
>
> ❷ 最常用配置选项:指定软件的安装路径
>
> ❸ 根据 `Makefile` 编译源代码
>
> ❹ 将编译完成的程序安装到系统中。通常需要 root权限
>
> ❺ 清除源代码目录中的编译结果

> [24] Windows 系统下的一些经典软件，如 ACDsee、Nero、Winamp 等，集成了越来越多的功能，使它们越来越臃肿。而且不能够只选择自己喜欢的功能，要么全盘接收，要么改寻它途
>
> [25] 通常发行版提供的安装包，已经进行了优化。自己编译的软件，性能未必更好

编译过程

将下面代码保存为 `Hello.c` ：

```
#include <stdio.h>
int main(void)
{
printf("Hello World!\n");
return 0;
}
```

> ❶ **printf()** 函数

执行命令 `cc Hello.c` [26]，得到一个可执行文件 `a.out` ，执行它 `./a.out`

可以看到，C的源代码(`Hello.c`)是纯文本，不能够直接执行。可执行代码是计算机的本机语言或机器语言表示的代码，这种语言是由数字代码表示的详细指令组成，不同的计算机具有不同的机器语言。

编译器是一个程序，其工作是将源代码转换为可执行代码。

- 编译器用来将 C语言 转换成特定的机器语言。
- 编译器还从C的库中向最终程序加入代码。[27]
- 编译器还检查源代码是否为有效的C语言程序。如果编译器发现错误，将报告错误，而且不生成可执行文件

编译器分三步完成这个工作：

预处理	调用预处理器 cpp 对源代码文件中的文件包含(include)、预编译语句(如宏定义 define 等) 进行分析
编译	调用编译器 cc 将源代码转换为中间代码
链接	调用链接器 ld 将中间代码与其它代码结合起来生成可执行文件

- 这种方法使用程序便于模块化。分别编译各个模块，然后使用链接器将编译过的模块结合起来。这样，如果需要改变一个模块，则不必重新编译所有其它模块。

可执行文件包含 `目标文件` 、 `库例程` 和 `启动代码`

编译器将源代码转换为机器语言代码(中间代码)，将结果放置在目标文件(`*.o`)中。虽然目标文件包含机器代码，但该文件还不能运行，它还不是一个完整的程序。

启动代码(start-up code)相当于程序和操作系统之间的接口。[28]

库例程为函数的实现。几乎所有C程序都利用标准C库中所包含的例程，目标代码文件不包含这一函数的代码，它只包含调用函数的指令。实际代码存储在一个称为“库”的文件中。库文件中包含许多函数的目标代码

链接器的作用是将这3个元素(目标代码、系统的标准启动代码和库代码[29])结合在一起，并将它们存放在可执行文件中。

> [26] 在 Linux 系统中，编译器为gcc，cc为它的链接
>
> [27] 库中包含许多标准例程供您使用，例如**printf()**。更准确的说，是一个被称为链接器(linker)的程序将库例程引入的，但在多数系统上，编译器为您运行链接器。
>
> [28] 硬件相同的情况下，在 DOS 或 Linux 下可以使用同样的目标代码，但 DOC 与 Linux 要使用不同的启动代码，因为这两种系统处理程序的方式是不同的。
>
> [29] 程序有两种方法来使用这些库函数，如果静态连接一个程序，这些函数就会被复制到可执行程序中,这就是 `lib*.a` 函数库的作用
>
> 如果你动态的连接一个程序（默认），那么当程序运行时需要库中的代码，它就会调用 `lib*.so` 中的内容。

gcc 编译器

gcc 是 GNU 推出的功能强大、性能优越的多平台编译器，是 GNU 的代表作品之一。它能将 C、C++语言源程序、汇编语言源程序和目标程序编译、链接成可执行文件，如果没有给出可执行文件的名字，gcc 将生成一个名为 `a.out` 的文件。

gcc 通过后缀来区分输入文件的类型：

后缀	类型
.c	C语言源代码文件
.a	由目标文件构成的档案库文件
.C|.cc|.cxx	C++源代码文件
.h	程序所包含的头文件
.i	预处理过的C源代码文件
.ii	预处理过的C++源代码文件
.m	Objective-C源代码文件
.o	编译后的目标文件
.s	汇编语言源代码文件
.S	预编译的汇编语言源代码文件

前面我们已经使用 gcc 编译了一个程序： `cc Hello.c`

gcc 还有许多选项：

-c	只编译，不链接成为可执行文件
-o 文件名	设定输出文件名。默认为 `a.out`
-g	加入调试符号(默认)。❶
-O	编译、链接时进行优化，耗时比较多，但产生的可执行文件执行效率更高
-O2	更高的优化级别，耗时更多

> ❶ 可以使用 gdb 进行调试使用下面的命令去掉调试符号：
>
> ```
> strip --strip-unneeded a.out
> strip --strip-debug a.out
> ```
>
> 不要在库文件上使用 **--strip-unneeded**

自动化编译

在前面的标准编译安装中，第一步是 `./configure` [30]，它会根据 `Makefile.in` 生成 `Makefile` 文件，然后make根据 `Makefile` 自动编译软件

通常在一个源码包中，已经包含了 `configure` 脚本和 `Makefile` 文件，作为课外知识，我们大致了解一下怎么生成这两个文件

autoconf

autoconf用来生成 `configure` 脚本，它可以检查系统特性、编译环境、环境变量、软件参数、依赖关系等

autoconf需要用到 m4

1. 用autoscan描源代码目录生成 `configure.scan` 文件
2. 将 `configure.scan` 改名为 `configure.in`
3. 用aclocal根据 `configure.in` 文件的内容，自动生成 `aclocal.m4` 文件
4. 使用autoconf，根据 `configure.in` 和 `aclocal.m4` 来产生 `configure` 文件

automake

automake可以从 `Makefile.am` 文件自动生成 `Makefile.in` ，它主要用来配置源代码

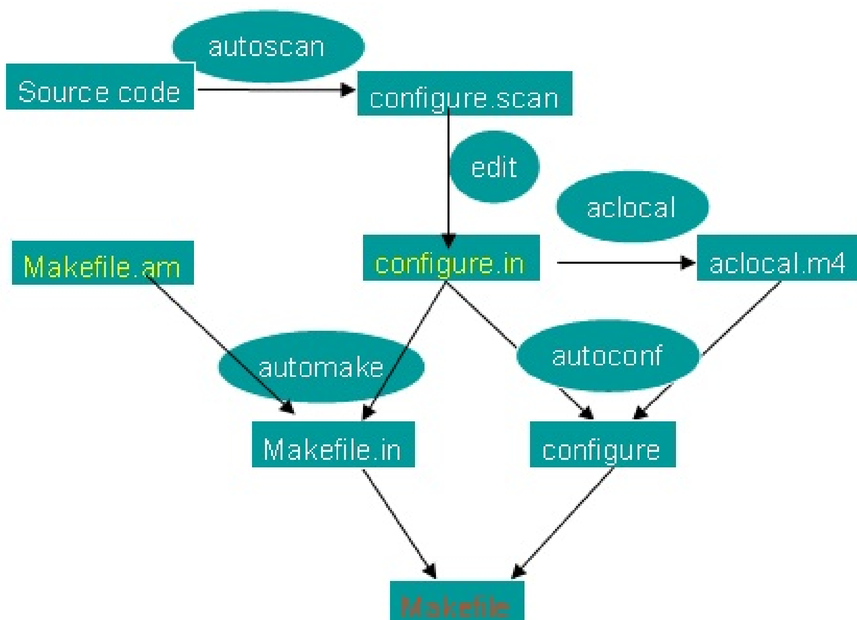
automake需用到perl

- 手工写 `Makefile.am`
- 使用 `automake` ，根据 `configure.in` 和 `Makefile.am` 来产生 `Makefile.in`

Makefile

使用 `configure` 脚本，配合 `Makefile.in` 可以生成 `Makefile` 文件，然后用make自动化的编译软件

这里有一张生成 `Makefile` 的流程图：



`Makefile` 的用途不只是编译软件，还可以利用它完成一些琐碎的工作，只要最后输出一个文件，都可以用make来完成

这是一个最简单的 `Makefile`

```
filelist:*
    ls -lF > filelist
```

> ❶ 输出的目标文件，不能省略。如果有多个文件，可以使用**all**
>
> ❷ 分隔符，不能省略
>
> ❸ 输入文件，可以省略
>
> ❹ 这一行必须以 `TAB` 字符起始，不能使用空格代替
>
> ❺ make的命令

可以使用变量代替命令，便于维护

```
TARGET = filelist
SOURCE =  *
ARG = -lF
APPLICATION = ls

$(TARGET):$(SOURCE)
    $(APPLICATION) $(ARG) $(SOURCE) > $(TARGET)
```

> ❶ 定义变量，传统上用大写
>
> ❷ 使用变量写 `Makefile`

`Makefile` 可以有多个目标文件，我们前面提到，gcc编译时先生成目标文件，再把目标文件链接成可执行文件，Makefile应该是这样的：

```
OBJECTS = main.o kbd.o command.o display.o \
              insert.o search.o files.o utils.o

exe : $(OBJECTS)
            cc -o exe $(OBJECTS)

main.o : main.c defs.h
        cc -c main.c
kbd.o : kbd.c defs.h command.h
        cc -c kbd.c
command.o : command.c defs.h command.h
        cc -c command.c
display.o : display.c defs.h buffer.h
        cc -c display.c
insert.o : insert.c defs.h buffer.h
        cc -c insert.c
search.o : search.c defs.h buffer.h
        cc -c search.c
files.o : files.c defs.h buffer.h command.h
        cc -c files.c
utils.o : utils.c defs.h
        cc -c utils.c
clean :
        rm exe $(OBJECTS)
```

> ❶ 如果写在多行，要用脱字符换行
>
> ❷ 如何生成中间文件
>
> ❸ 伪目标文件，`make clean` 并不生成 `clean` 文件，而是清理编译结果

`Makefile` 还有很多强大的机制，这里就不详细介绍了

> [30] 执行当前目录下的 `configure` 脚本

使用 make

利用 `configure` 所产生的 `Makefile` 文件有几个预先设定的目标可供使用：

目标	用途
make all	产生预设的目标，只敲入 `make` 也可以
make clean	清除编译结果
make distclean	除了清除编译结果，也把configure所产生的 `Makefile` 清除掉
make install	将程序安装到系统中
make dist	将程序和相关的文档打包为一个压缩文档以供发布
make distcheck	打包并检验

emerge

虽然我们能够使用autoconf、automake、make等工具实现自动化编译，但这种针对单个软件包的编译系统，在编译多个软件时仍然十分繁琐

假设需要编译emacs和vim，使用 xft字体、图形界面支持，并去掉调试符号，需要分别作如下配置

```
emacs
./configure --prefix=/usr/local/     \
`--no-debug`                \
`--with-xft  --with-x-toolkit=gtk`        \
--with-freetype
vim
./configure --prefix=/usr/local/     \
`--no-debug`                \
`--with-xft --with-x-toolkit=gtk`
```

这就像点菜时，你必须告诉厨师：鱼香肉丝(多放辣椒、不放蒜)、宫保鸡丁(多放辣椒、不放蒜)、麻婆豆腐(多放辣椒、不放蒜)……

实际上，大多数人这样点菜：鱼香肉丝、宫保鸡丁、麻婆豆腐……多放辣椒、不放蒜

emerge就是这样一种点菜方式，它是gentoo的包管理系统，提供了更为现代化的编译方式

可以通过指定USE标记 `xft` 、 `gtk` 、 `-debug` 来确定所有软件的编译方式

设置 USE标记

以下方法按优先级由低到高排列：

`/etc/make.profile/` 目录是一个符号链接，里面包含一些 `make.defaults` 文件，放置开发者设置的 USE标记[31]：

```
/usr/portage/profile/base/make.defaults
/usr/portage/profile/default-linux/make.defaults
/usr/portage/profile/default-linux/x86/make.defaults
/usr/portage/profile/default-linux/x86/2008.0/make.defaults
```

在 `/etc/make.conf` 文件中声明永久 USE标记(推荐)

```
USE="nptl nptlonly nls cjk php mysql -kde -qt3 -qt4"
```

- 带 `-` 的 USE标记，表示排除

在 `/etc/portage/package.use` 文件中为单个包声明 USE标记

```
app-editors/emacs-cvs xft
www-servers/lighttpd fastcgi
dev-lang/php mysqli cgi gd ctype pcre session unicode pic posix
dev-db/phpmyadmin vhosts
app-shells/zsh doc
net-ftp/pure-ftpd -ldap mysql pam ssl vchroot
```

使用环境变量声明临时 USE标记

```
USE="-java"
emerge seamonkey
```

查看使用的 USE标记：

```
merge --pretend --verbose seamonkey

Calculating dependencies ...done!
[ebuild   R   ] www-client/seamonkey-1.0.7  USE="crypt gnome java -debug -ipv6
-ldap -mozcalendar -mozdevelop -moznocompose -moznoirc -moznomail -moznopango
-moznoroaming -postgres -xinerama -xprint" 0 kB
```

编译选项

`/etc/make.conf`

```
CFLAGS="-O2 -march=i686 -pipe"
CXXFLAGS="-O2 -march=i686 -pipe"

CHOST="i686-pc-linux-gnu"
MAKEOPTS="-j2"

FEATURES="parallel-fetch ccache"
CCACHE_DIR="/var/tmp/ccache"
CCACHE_SIZE="2G"

ACCEPT_KEYWORDS="x86"

USE="nptl nptlonly nls cjk php mysql"

FETCHCOMMAND="/usr/bin/axel -a -n4 \${URI} -o \${DISTDIR}"
RESUMECOMMAND="/usr/bin/axel -a -n4 \${URI} -o \${DISTDIR}"
```

> ❶ 针对C语言的优化选项，**-march=**设置目标架构
>
> ❷ 针对C++语言的优化选项
>
> ❸ 进行编译工作的机器架构
>
> ❹ 编译选项
>
> ❺ emerge 特性。并行下载、使用 ccache 缓冲编译结果
>
> ❻ ccache 缓存目录
>
> ❼ ccache 缓存大小
>
> ❽ 通过关键字选择分支。`x86`表示 x86 架构的稳定分支，`~x86`表示 x86 架构的不稳定分支
>
> ❾ USE标记
>
> ❿ 使用axel加速下载

gentoo支持多种架构：x86、 sparc、 amd64、 ppc、 ppc64、 alpha、 hppa、 mips、 ia64、 arm，我们使用的 PC 多为 x86 架构

假设你主要使用 x86 稳定分支，但少数软件要使用最新版本，在`/etc/portage/package.keywords`文件中为单个包设置关键字

```
app-editors/emacs-cvs ~x86
x11-misc/emacs-desktop ~x86
app-i18n/fcitx ~x86
app-editors/vim x86
app-editors/vim-core x86

media-video/mplayer ~x86
media-libs/win32codecs ~x86

app-i18n/man-pages-zh_CN ~x86
```

微调

在`/etc/portage/`目标下包含一些文件，可以在软件包级别上进行调节。前面已经介绍了`package.use`和`package.keywords`

package.keywords

还未被确认适合你的系统或架构，但是你希望能安装的软件包

package.use

特定软件包而不是整个系统使用的 USE标记

package.provided

屏蔽的软件包(需要指明版本号)

package.mask

永远不希望 Portage 安装的软件包。

package.unmask

被 Gentoo 开发者屏蔽的软件包，但是你希望能安装的软件包。

软件包可能由于以下原因被屏蔽

~架构 keyword	意味着这个软件没有经过充分的测试，不能进入稳定分支，请等待一段时间后在尝试使用它
-架构 keyword 或 -* keyword	意味着这个软件不能工作在您机器的体系结构中
missing keyword	意味着这个软件还没有在您机器的体系结构中进行过测试
package.mask	意味着这个软件被认为是损坏的，不稳定的或者有更严重的问题，它被故意标识为“不应使用”
profile	意味着这个软件不适用于您的 profile。安装这样的应用软件可能会破坏您的系统，或者只是不能与您使用的 profile 相兼容

例如：

```
gnome-base/gnome-2.8.0_pre1 (masked by: ~x86 keyword)
lm-sensors/lm-sensors-2.8.7 (masked by: -sparc keyword)
sys-libs/glibc-2.3.4.20040808 (masked by: -* keyword)
dev-util/cvsd-1.0.2 (masked by: missing keyword)
games-fps/unreal-tournament-451 (masked by: package.mask)
sys-libs/glibc-2.3.2-r11 (masked by: profile)
```

使用 emerge

> 注意：本部分内容来源于gentoo 中文手册

查找.

查找名字包含 pdf 的软件包

```
emerge --search pdf
```

查找与 pdf 相关的软件包

```
emerge --searchdesc pdf
emerge -S pdf
```

查看软件拥有的USE标记

```
emerge -vp 软件包名称
```

管理.

安装软件包

```
emerge 软件包名称
```

模拟安装软件包

```
emerge --pretend 软件包名称
```

下载软件包的源代码包

```
emerge --fetchonly 软件包名称
```

从系统中删除软件包

```
emerge --unmerge 软件包名称
```

更新.

更新系统

```
emerge --update --ask world
emerge -ua world
```

更新整个系统

```
emerge --update --deep world
emerge -uD world
```

使用新的USE标记 重新构建系统

```
emerge --update --deep --newuse world
emerge -uDN world
```

移除孤立依赖的软件包

```
emerge --update --deep --newuse world
emerge --depclean
revdep-rebuild
```

> ❶ 重新构建系统
>
> ❷ 清除孤立依赖包
>
> ❸ 重新构建依赖关系

revdep-rebuild工具由gentoolkit包提供；使用前别忘了首先 emerge 它：

```
emerge gentoolkit
```

> [31] 在升级 Portage 的时候，这些文件将会被覆盖，请不要在这里设置

第 19 章 图形界面

目录

简介

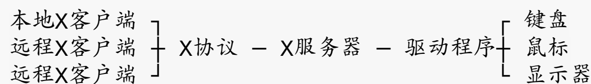
Xwindow 是工作站图形系统的工业标准，它有多种不同的实现，多数 Linux 系统中使用 Xorg。

当然，Xwindow 有悠久的历史和传统，不过那不在我们讨论的范围。您要注意的有两点：

- Xwindow 和 Xbox 中的“X”意义是不同的， X 只是 W 之后的一个字母，差不多应该这样理解，Xwindow 是 Window 的接班人 （注意，Window 不是 Windows）
- 同样，也不要把 Xwindow 说成是 Xwindow `s` ，那是一种亵渎！一切伟大的创造，都应得到应有的理解和尊重

架构及原理

Xwindow 使用服务器—客户端架构。无论本地图形界面，还是远程图形界面，都以同样的流程工作。这样便不需要分别进行设计和维护。



Xserver

Xwindow 系统服务器端，通过驱动程序（硬件规范）来管理硬件资源。

例如：当我们移动鼠标时，通过驱动程序[32]，向 Xserver 发送信息：

```
“向右移动200点，向上移动100点”（向右上移动）；“按下左键”……
```

Xserver 作出如下响应：

1. 上一次鼠标停止的坐标为 600,500
2. 向右 200，向上 100。现在鼠标位于坐标 800,600
3. 坐标 800,600 处，为窗口 Firefox 的“关闭”按钮
4. 根据预设动作，将 “点击 Firefox 窗口的关闭按钮” 翻译为 “关闭窗口 Firefox”
5. 向X客户端 Firefox 发送一个“退出”消息
6. Xserver 通过显示子系统（显卡、显示器），全程显示鼠标的位置和移动
7. 事实上，向程序发送“退出”信号，通常窗口管理器完成。为了描述方便，这里暂不区分。稍后，我们将向您介绍 窗口管理器 的其它一些细节。

Xclient

Xwindow 系统客户端，通过 X协议，实现与 Xserver 的交互。例如：

1. Xclient （假设 Firefox） 接收 Xserver 的消息： 输入焦点在地址栏的范围内，“linuxtoy.org”，回车
2. Firefox 根据预设动作，将这些消息识别为 “打开链接 linuxtoy.org”
3. Firefox 向域名服务器请求 链接 “linuxtoy.org”。域名服务器将这个请求转换为 “http://linuxtoy.org/” 和 IP地址211.148.131.7，发送回 Firefox
4. Firefox 将 “http://linuxtoy.org/”显示在地址栏（向 Xserver 发送请求，在地址栏位置显示这个地址）
5. Firefox 向地址 211.148.131.7 请示显示页面
6. Firefox 将服务器发送回的页面显示在主窗口中

Xprotocol

Xwindow 系统协议，Xserver 和 Xclient 之间进行通信的规则

> [32] 大多数的鼠标不需要专门的驱动程序，因为它们符合某一硬件规范，例如：有四个移动方向和三个键

窗口管理器

Window Manager 是一种特殊的 Xclient。

使用窗口管理器时，Xserver 并不直接与其它 Xclient 通信，而是通过 WM 中转，当一些消息被定义为 WM 指令时，它们会被拦截。例如 Alt+F4 关闭窗口、拖动标题栏......

消息“打开链接 linuxtoy.org”，具体内容如下：

```
输入焦点在地址栏的范围内，“linuxtoy.org”，回车
```

Xserver 并不能直接判断焦点，而是这样：

1. Xserver 向 WM 发送位置和点击的信息，WM 根据当前的“焦点策略”确定激活（最上层）的窗口为 Firefox
2. Xserver 将 Firefox 显示在最上层，高亮显示它的标题栏
3. 在窗口 Firefox 内点击地址栏，或者 Ctrl+L，Xserver 将位置信息发送给 WM，WM 再发送给 Firefox
4. Firefox 判断当前焦点后，显示一个闪动的文字输入光标
5. Firefox 将输入光标通过 WM 发送给 Xserver，Xserver 在屏幕相应位置进行显示

那么，“窗口管理器”到底能作些什么呢？其实它所作的只有一件事——管理窗口。例如：

- 最上层的窗口会把其它窗口挡住

- 它通常是一个“已激活窗口”，根据不同的“焦点策略”，窗口管理器确定被激活的窗口。

  激活窗口标题栏高亮显示，接收大部分的键盘消息和窗口内的鼠标点击消息。

- 为了美观和容易分辨，大多数窗口都要有标题栏和边框。

  为了方便，标题栏上还要有一些按钮，比如：最小化，最大化，关闭（这些按钮是窗口管理器请求的小窗口）

- 一个窗口可以在另一个窗口旁边显示，而不一定完全被遮挡。为了实现这一点，就要控制窗口显示的位置

- 为了控制窗口的显示位置，需要将整个屏幕用座标描述，最好的办法是绘制一个填充整个屏幕的窗口，也就是根窗口。
- 因为根窗口是最大的，所以它可以严严实实的遮挡任何窗口，为了避免这一点，根窗口永远在最底层。这很形象的说明了为什么它叫作"根窗口" ……root
- 根窗口不一定只有一个，大多数的窗口管理器可以使用 "工作区" ，来切换显示多个根窗口
- 根窗口固定位置上通常放置一些其它 Xclient 的窗口，例如底部面板，顶部面板，侧面板，程序启动图标
- 面板上又可以放一些其它的 Xclient 窗口，如任务条，启动栏，菜单……

  任务条可以以图标显示正在运行的任务，还可以作其它的杂活，像自动挂载 USB设备……

启动流程

我们知道 init 是 linux 的根进程，是所有进程的父进程。同样， xinit 是所有 Xwindow 进程的根进程

Startx

`startx` 可以在命令行下启动图形界面。执行**startx** 命令时，实际执行这一命令：

```
xinit /etc/X11/xinit/xinitrc -- /etc/X11/xinit/xserverrc
```

根据脚本 `/etc/X11/xinit/xserverrc` 启动 Xserver，同时根据脚本 `/etc/X11/xinit/xinitrc` 启动指定 Xclient 进程，例如窗口管理器

`/etc/X11/xinit/xserverrc` 以预设的参数运行程序 /usr/bin/X11/X

`/etc/X11/xinit/xinitrc` 脚本则指向 `/etc/X11/Xsession` ，依次启动 `/etc/X11/Xsession.d` 目录中的脚本

- 您可以在用户配置文件 `~/.Xsession` 中定义使用的 WM，它的优先级高于全局配置文件 (对于 GDM会话 不起作用)
- startx 启动时，并不会再进行身份认证。因为它启动的是 /etc/X11/Xsession.d/gnome-session ，而不是 GDM会话

GDM会话

许多 Linux 系统启动时自动进入图形界面，不需要运行 **startx** 命令

在某些启动级别中，包含了 gdm 的启动脚本，例如 ： `/etc/rc2.d/S13gdm`

1. 指向 `/etc/gdm/gdm-cdd.conf` 文件，加载预设视觉主题，启动 `/usr/lib/gdm/gdmgreeter` （登录屏幕）
2. 用户身份认证完成后，启动 `/etc/X11/default-display-manager` 这个文件中设定的默认窗口管理器 /usr/sbin/gdm

gdm 在启动时，会要求用户名和密码，也就是我们看到的登录屏幕（gdmgreeter）

- `/usr/share/xsessions` 目录下为所有可用登录会话的脚本

配置文件

X服务器

X服务器的主要配置文件为 `/etc/X11/xorg.conf`

```
Section "ServerLayout"                #布局
Identifier      "Xorg Configured"     #布局标识
Screen     0   "Default Screen" 0 0   #屏幕标识
InputDevice    "Generic Keyboard"     #键盘标识
InputDevice    "Configured Mouse"     #鼠标标识
EndSection

Section "Module"                      #模块
        Load  "ddc"
        Load  "dbe"
        Load  "dri"
        Load  "extmod"
        Load  "glx"
        Load  "bitmap"
        Load  "type1"
        Load  "freetype"
        Load  "record"
EndSection

Section "Files"                       #X核心字体路径
        FontPath     "/usr/share/X11/fonts/75dpi"
        FontPath     "/usr/share/X11/fonts/100dpi"
        FontPath     "/usr/share/X11/fonts/misc"
        FontPath     "/usr/share/X11/fonts/cyrillic"
        FontPath     "/usr/share/X11/fonts/100dpi/:unscaled"
        FontPath     "/usr/share/X11/fonts/75dpi/:unscaled"
        FontPath     "/usr/share/X11/fonts/Type1"
        FontPath     "/usr/share/fonts/Chinese/wqy-bitmapfont"
EndSection

Section "Screen"                      #屏幕
    Identifier "Default Screen"
    Device     "Card0"          #指定显卡
        Monitor    "Monitor0"         #指定显示器
    DefaultDepth     24           #默认色深为24
    SubSection "Display"          #可用分辨率
    Depth     24              #24位色深下可用分辨率
    Modes    "1280x1024" "1152x864" "1024x768" "800x600" "720x400" "640x480"
    ………………
    EndSubSection
EndSection

Section "Device"                       #显卡
    Identifier  "Card0"
        Driver      "vesa"           #驱动
    VendorName  "All"
    BoardName   "All"
EndSection
```

> ❶ 默认分辨率为默认色深下的第一个分辨率

配置文件内部结构

```
/
├/ "ServerLayout"                布局
│├ "InputDevice" keyboard        键盘
│├ "InputDevice" mouse           鼠标
││
│└/ "Screen"                     显示子系统
│ ├ "Monitor"                    显示器
│ ├ "Device"  videocard          显卡
│
│
├ "Files"                         字体
└ "Module"                        模块
```

X客户端

在 `/etc/X11/Xsession` 文件中可以发现下列内容

```
OPTIONFILE=/etc/X11/Xsession.options

SYSRESOURCES=/etc/X11/Xresources
USRRESOURCES=$HOME/.Xresources

SYSSESSIONDIR=/etc/X11/Xsession.d
USERXSESSION=$HOME/.xsession
ALTUSERXSESSION=$HOME/.Xsession
ERRFILE=$HOME/.xsession-errors
```

> ❶ 设定X进程的启动参数。例如允许用户进程 allow-user-xsession
>
> ❷ X资源文件。许多程序保留了X接口，允许X服务器管理一些视觉选项，例如窗口内的字体，配色等
>
> ❸ X进程。可以设置一些启动时自动运行的程序，也可以用来设定自己的窗口管理器（窗口管理器和桌面环境或者登录管理器是无关的）

字体

freetype 渲染引擎

作为 Xorg 服务器的一个模块，freetype 的功能包括读取 Truetype 字体信息，如大小、分辨率、编码等，并以之为依据渲染字体 - freetype2.x 相对于 freetype1.x 增加了抗锯齿等功能 - (`/etc/X11/xorg.conf` 的 Module 字段中，可以选择字体渲染模块)

freetype 只负责渲染字体。而查找字体，则可以由 X服务器、X客户端 或者 字体服务器来 完成。找到字体后，使用 freetype 引擎进行渲染

X核心字体

X服务器根据X客户端的请求（字符编码），查找字体并进行渲染，然后显示

Xft字体

X客户端自行查找字体并进行渲染，X服务器只负责显示。由于 Xft字体 的渲染在客户端完成，所以它可以动态的加载，而不需要随同X服务器一同启动

字体服务器

当客户端请求字体时，X服务器将请求转发到字体服务器，由字体服务器查找字体，并使用 freetype 引擎渲染，将结果传回X服务器，X服务器进行显示

X核心字体

`/etc/X11/xorg.conf` 中可以配置X核心字体的搜索路径

```
Section "Files"
FontPath "/usr/X11R6/lib/X11/fonts/misc/"
FontPath "/usr/X11R6/lib/X11/fonts/Type1/"
FontPath "/usr/X11R6/lib/X11/fonts/Speedo/"
FontPath "/usr/X11R6/lib/X11/fonts/100dpi/"
FontPath "/usr/X11R6/lib/X11/fonts/75dpi/"
EndSection
```

- 当X客户端向X服务器请求显示文字的时候，X服务器会按上面列表的先后顺序查找字体。显示中文时，如果第一个路径中的字体不包含中文，则查找下面的路径，直到发现中文字体
- 请将您偏好的字体放在靠前的位置

要使安装的字体能够作为X核心字体使用，将字体的安装路径添加到上面的列表中，使用 mkfontscale 、 mkfontdir 扫瞄文件夹中的字体，并生成索引，就可以了（建议使用 ttmkfdir 生成 `fonts.scale` ，将其复制为 `fonts.dir` ）

字体的选择及显示风格，可以修改 GTK1 的配置文件，或者在 `Xresources` 文件中对程序单独进行定义

事实上，在我们的日常应用中，X核心字体环境并不常见，使用 GTK1 图形库的程序、某些类型的终端......

XFT字体

Xft字体相关选项在 `/etc/fonts/fonts.conf` 文件中配置

```
<dir>/usr/share/X11/fonts</dir>
<dir>/usr/share/fonts</dir>
<dir>/usr/local/share/fonts</dir>
<dir>~/.fonts</dir>
```

可以使用 **fc-cache** 命令，递归扫瞄上面目录中的字体（包括子文件夹中的字体），建立字体缓存

多数支持 GTK2 或者 Qt 图形库的X客户端能够使用 Xft字体 渲染技术[33]

安装字体，只要将字体拷贝到以上任意目录， **fc-cache -fv** 刷新字体缓存即可 （选项: **-f** 强制刷新; **-v** 显示过程）

使用命令 **fc-list** 列出所有可用字体。字体的选择及显示风格，可以修改 GTK2 或者 Qt 的配置文件，建议使用图形界面配置[34]

> [33] GTK2 为 Gnome 使用的图形库，Qt 为 KDE 使用的图形库。相对来说，GTK2 图形库在程序的 GUI 设计中更加通用
>
> [34] 一般情况下，桌面环境中附带了相关程序，例如 gnome-font-properties

第 20 章 国际化

目录

I18N 简介

习惯上，`internationalization`（国际化）简写为`I18N`,中间的数字为省略的字母个数。

在 Linux 系统中，通过定义一组环境变量来设置程序的语言环境，以实现对`I18N`的支持，这种机制称为：`locale`（本地化）

locale 变量

LANG	低优先级全局 locale 变量	如果下面的变量未赋值，默认使用此变量的值
LC_COLLATE	比较和排序习惯	会影响目录列表的分类显示等
LC_CTYPE	定义系统的字符处理特性	哪些字符能被视为字母、数字，等等；与中文输入关系密切
LC_MESSAGES	提示信息,错误信息, 状态信息, 标题, 标签, 按钮和菜单等	
LC_MONETARY	定义货币单位和货币型数值的格式	
LC_NUMERIC	定义非货币型数值的格式	影响到千位分隔符和小数分隔符等
LC_TIME	定义日期和时间的格式	
LC_NAME	姓名书写方式	
LC_ADDRESS	地址书写方式	
LC_TELEPHONE	电话号码书写方式	
LC_MEASUREMENT	度量衡表达方式	
LC_PAPER	定义默认的纸张尺寸	
LC_IDENTIFICATION	对 locale 自身包含信息的概述	
LC_ALL	高优先级全局 locale 变量	为此变量赋值会强行覆盖上面变量的值，不推荐

- 假如未设置以上变量，系统将采用 POSIX 作为 lcoale，也就是 C locale

locale 值

`locale` 变量的值有三个要素：语言代码 (Language Code)、 地域代码 (Country Code) 和字符集(Encoding)[35]：

```
语言代码[_地域代码[.字符集]]
```

例如：

语言/国家代码	描述
en_US.ISO-8859-1	美国英语
en_US.UTF-8	
zh_CN.UTF-8	简体中文
zh_TW.UTF-8	繁体中文

字符集

众所周知，计算机中的信息，是以数字形式表示的，字符也不例外。字符以数字编号的形式存储，使用时，根据这个编号，在字符集中找到相应的字符

字符集是字符在系统内的编码方式，也就是通常所说的内码[36]。不同的字符集有不同的编码方式，

例如“码”字，它的 GB2312编码 为 426B；UTF-8编码 为 E7A081 。如果错误的以 GB2312编码 来检索 E7A081，将会产生类似 “锘跨爜” 的乱码

只要系统中安装了中文字体，通过字符集的支持，便可以正常显示中文，而无需设置 locale；locale 可以使操作界面显示中文，并可以使用中文输入等

其它

另外还有一个本地化变量叫做 LINGUAS。它会影响到基于 gettext 的程序[37]；它还能决定某些特殊软件包的本地化，比如kde-i18n和openoffice。这个变量的值为一组以空格分隔的语言代码：

```
LINGUAS="zh en"
```

> [35] 中文 Windows 系统中使用的字符集为 `GB2132` ，而 `UTF-8` 是大势所趋，它能够显示比 GB2312 更多的字符
>
> [36] 参见 `/usr/share/i18n/charmaps` 文件
>
> [37] 通过编译时安装本地化文件的方式实现 `I18N` ，编译时需要Native Language Support(NLS)支持

设置 locale

设置 `locale` 其实就是设置环境变量。在 `/etc/environment` [38]文件中设置全局环境变量：

```
LANG="zh_CN.UTF-8"
```

在 `~/.profile` 或 `~/.bashrc` 中设置用户环境变量：

```
export LANG="zh_CN.UTF-8"
```

完成后可以使用**locale**命令检验

生成 locale

如果你设置了一个不可用的 locale，请使用 localedef 生成该 locale

```
localedef -c -i en_US -f ISO-8859-15 en_US.ISO-8859-15
```

可能你在系统中只要用到一个或者两个 locale。你可以在 `/etc/locale.gen` 中指定所需的的 locale。添加 locale 到 `/etc/locale.gen` ：

```
en_GB ISO-8859-1
en_GB.UTF-8 UTF-8
de_DE ISO-8859-1
de_DE@euro ISO-8859-15
```

下一步是执行locale-gen。它会生成 `/etc/locale.gen` 文件中指定的所有 locale。

> [38] 在 gentoo 中使用专门的文件 `/etc/env.d/02locale` 设置 locale

locale 策略

中文图形界面

由于 Linux 的控制台不能方便的显示中文，所以最实用的方案是“英文控制台＋中文图形界面”

为了能够正常处理中文，需要使用locale-gen生成中文 locale，在 `/etc/locale.gen` 文件中添加如下内容：

```
zh_CN.UTF-8 UTF-8
zh_CN.GB18030 GB18030
zh_CN.GBK GBK
zh_CN GB2312
```

然后在 gdm 启动菜单中选择中文，或者写入配置文件 `~/.dmrc`

```
[Desktop]
Session=openbox
Language=zh_CN.UTF-8
```

英文界面+中文输入

使用如下设定：

```
LANG="en_US.UTF-8"
LC_CTYPE="zh_CN.UTF-8"
```

关于输入法的设置

在 `~/.profile` 或 `~/.bashrc` 中设置用户环境变量：

例 20.1. 输入法配置 **`.profile`**

```
export XMODIFIERS="@im=fcitx"
export GTK_IM_MODULE=xim
export QT_IM_MODULE=xim
fcitx&
```

> ❶ 使用fcitx输入法
>
> ❷ GTK2 程序输入法引擎。fcitx 使用xim引擎
>
> ❸ QT 输入法引擎。

第 21 章 内核

目录

简介

内核是系统的引擎，它是一个系统运行起来的先决条件。

- 内核管理硬件，是程序和硬件之间的接口
- 内核对进程进行调度，将硬件资源分配给不同任务，使系统可以同时运行多个任务
- 内核对内存进行管理，将内存空间分配给任务，将使用不频繁的页面转移到交换分区
- 内核还管理文件系统，进程间通信和网络

内核包含几个重要的子系统：

进程调度（SCHED）

控制进程对CPU的访问。当需要选择下一个进程运行时，由调度程序选择最值得运行的进程。可运行进程实际上是仅等待CPU资源的进程，如果某个进程在等待其它资源，则该进程是不可运行进程。Linux使用了比较简单的基于优先级的进程调度算法选择新的进程。

内存管理（MM）

允许多个进程安全的共享主内存区域。Linux的内存管理支持虚拟内存，即在计算机中运行的程序，其代码，数据，堆栈的总量可以超过实际内存的大小，操作系统只是把当前使用的程序块保留在内存中，其余的程序块则保留在磁盘中。必要时，操作系统负责在磁盘和内存间交换程序块。内存管理从逻辑上分为硬件无关部分和硬件有关部分。硬件无关部分提供了进程的映射和逻辑内存的对换；硬件相关的部分为内存管理硬件提供了虚拟接口。

虚拟文件系统（VirtualFileSystem,VFS）

隐藏了各种硬件的具体细节，为所有的设备提供了统一的接口，VFS提供了多达数十种不同的文件系统。虚拟文件系统可以分为逻辑文件系统和设备驱动程序。逻辑文件系统指Linux所支持的文件系统，如ext2,fat等，设备驱动程序指为每一种硬件控制器所编写的设备驱动程序模块

网络接口（NET）

提供了对各种网络标准的存取和各种网络硬件的支持。网络接口可分为网络协议和网络驱动程序。网络协议部分负责实现每一种可能的网络传输协议。网络设备驱动程序负责与硬件设备通讯，每一种可能的硬件设备都有相应的设备驱动程序。

进程间通讯(IPC)

支持进程间各种通信机制。

处于中心位置的进程调度，所有其它的子系统都依赖它，因为每个子系统都需要挂起或恢复进程。一般情况下，当一个进程等待硬件操作完成时，它被挂起；当操作眞正完成时，进程被恢复执行。例如，当一个进程通过网络发送一条消息时，网络接口需要挂起发送进程，直到硬件成功地完成消息的发送，当消息被成功的发送出去以后，网络接口给进程返回一个代码，表示操作的成功或失败。其他子系统以相似的理由依赖于进程调度。

启动流程

系统启动时，引导管理器首先加载内核，内核被载入后执行以下操作：

- 内核自解压* 初始化阶段
- init 阶段。init 进程启动后，执行当前运行级别的脚本：

调整内核

module-init-tools

编译内核

第 22 章 Grub

目录

硬件基础

一块硬盘，它起始的一部分扇区为主引导扇区，包括 MBR（主引导纪录）和 DPT（分区表，您可以阅读“分区概念”一节中相关内容）

每个分区起始的一部分扇区，为分区引导扇区。

在分区引导扇区之后的部分，为文件系统的索引，文件系统通过它定位文件在硬盘上的位置。不同的文件系统采用不同的索引，例如 FAT 文件系统使用文件分配表和目录区。

绝大多数操作系统，对硬盘的读写操作，通过文件系统来完成，因此引导扇区中的内容，我们不能够在文件系统中进行操作，而需要专用软件，比如引导管理器。

我们对文件进行修改后，操作系统会将文件系统索引中的内容同步。

系统引导流程

1. 系统启动时，首先引导至 MBR，将控制权移交安装在 MBR 中的引导管理器❹
2. 引导管理器装载自身[39]
3. 引导管理器读取分区中的配置文件，并按配置文件中预设的参数运行❺

4. 引导管理器根据您的选择，可能会有如下活动
    - 加载内核，启动 Linux 系统检查活动分区，并引导它
    - Windows 系统:读取相应分区的引导扇区，将控制权移交该扇区中的引导管理器

> ❹ Windows 使用 NTLDR，Linux 通常用 Grub
>
> ❺ 例如，Grub 读取“`/boot/grub/menu.lst`”文件中内容，将可引导系统通过菜单显示

Linux 系统在安装 Grub 时，会提问您安装在 MBR 或者分区引导扇区中。如果将 Grub 安装在分区引导纪录中，您必须确保 MBR 中的引导管理器能够正确的引导至分区引导扇区。

如果您在 MBR 中使用的是 Windows 的引导管理器 NTLDR，完成这件工作会非常困难，因而我们推荐您使用 Grub。

> [39] Grub 先装载 MBR 中的 stage1，通过 stage1 来装载文件系统中的 stage2，显示菜单和 Shell 等待用户。
>
> 有时 stage1 不能识别 stage2 所在分区的文件系统，这就需要装载 stage1.5 来连接 stage1 和 stage2

Grub 介绍

Grub 主要有以下功能：

- 菜单式选择
- 命令行模式
- 支持开机画面
- 支持大硬盘

其它的功能还有很多，就不一一介绍了。

您可以运行命令 **grub** 启动它。会显示一些版本信息和使用提示，当然还有命令提示符，如下：

```
GNU GRUB  version 0.97  (640K lower / 3072K upper memory)

   [ Minimal BASH-like line editing is supported.   For
   the   first   word,  TAB  lists  possible  command
   completions.  Anywhere else TAB lists the possible
   completions of a device/filename. ]

grub>
```

您可以使用 `TAB` 键补全命令和路径，这非常重要，因为 Grub 中路径表示方式与操作系统是不同的，您可能比较陌生，所以尽量用 TAB 补全它，既方便，也不容易出错。

您可以在 *grub>* 提示符后按 `TAB` 键，会将所有可用的命令显示出来。

呵呵，是不是有点晕，命令可真不少啊！！！其实我们会用到的命令只有两个:**root setup**

Grub 术语

在分区概念章节里，我们已经介绍了 Linux 系统中表示分区的方法 `/dev/sda5`

```
/dev/sdMN       M 为 a 起始的小写字母，表示硬盘序号；N 为 1 起始的数字，表示分区序号
```

Grub 中使用的表示方法为 `hd0,1`

```
hdX,Y           X 为 0 起始的数字，表示硬盘序号；Y 为 0 起始的数字，表示分区序号
```

留意它们之间的区别：

- N 从 1 开始计数，X 和 Y从 0 开始计数
- N 为 1~4，它是一个主分区；N 为 5 或大于 5，它是第(N-4)个逻辑分区。Y 按分区在硬盘上排列的顺序排列，无论它表示的是主分区还是逻辑分区。

举例来说：

分区分布	主分区	主分区	逻辑分区	逻辑分区	主分区
/dev/sdMN	sda1	sda2	sda5	sda6	sda3
hdX,Y	hd0,0	hd0,1	hd0,2	hd0,3	hd0,4

现在我们来看 **root** 和 **setup** 命令的使用，`sudo grub` 进入 Grub 交互模式：

```
grub>root (hd0,1)
grub>setup (hd0)
```

> ❶ 这个命令将 Grub 的根分区定位为 “(hd0,1)”
>
> ❷ 这个命令表示将 Grub 安装在“(hd0)”，因为没有指定安装的分区，所以安装位置为 MBR

Grub 的根分区

为 Grub 配置文件 `grub/menu.lst` 和 `stage[[39](ch22s02.html#ftn.grub-stage)]` 文件所在分区。假如您单独为 `/boot` 目录挂载了一个分区，那么 Grub 的根分区通常为您系统中 `/boot` 目录所在的分区。

搞错了根分区，Grub 就不能正确读取配置文件，自然不能正确引导。[40]

系统根目录所在分区

Linux 根目录 `/` 的挂载分区。Linux 系统的分区挂载信息保存在文件系统分配表 `/etc/fstab` 文件中

Grub 首先读取根分区[41]中的 `/boot/grub/menu.lst` 文件，并转到引导分区，如果是 Windows 等系统，则将控制权移动分区引导扇区中的启动管理器。如果是 Linux 系统，则加载内核和设备，并根据 `/etc/fstab` 文件的内容挂载文件系统。

看这个例子：（假设 Grub 安装在 MBR 中，Grub 的安装位置为 `(hd0)`）

```
title       Linux
root        (hd0,0)
kernel      (hd0,1)/vmlinuz-2.6.15-25-686 root=/dev/sda3 ro splash vga=0x31b
initrd      (hd0,1)/initrd.img-2.6.15-25-686
boot
```

> ❶ 这一行表示引导分区为第一块硬盘的第一个分区 (hd0,0)。如果下面也指定了这个参数，那么下面的优先
>
> ❷ 表示当前系统的 /boot 目录挂载到第一块硬盘的第二个分区 (hd0,1)，它是引导分区。一般情况下在 “root”项中指定
>
> ❸ 表示当前系统的 / 目录挂载到第一块硬盘的第三个分区 (hd0,2) ，内核根据该分区中的 `/etc/fstab` 文件来挂载文件系统
>
> ❹ 同2

> [40] Windows 等系统的引导分区为它的安装分区，Linux 系统的引导分区为它的 `/boot` 目录所在的分区，也就是 Grub 根分区
>
> [41] 配置文件中并不能指定 Grub 根分区，它要在 Grub 安装过程中指定

Grub 配置文件

`/boot/grub/menu.lst` 文件，主要由一些下面这样的块构成的

```
timeout   5
default   0

title       Linux
root        (hd0,2)
kernel      (hd0,2)/boot/vmlinuz-2.6.15-25-686 root=/dev/sda3 ro splash vga=0x31b
initrd      (hd0,2)/boot/initrd.img-2.6.15-25-686
boot

title       Windows
root        (hd0,0)
makeactive
chainloader +1
```

> ❶ 等待时间，如果用户在这段时间内未进行选择，则引导默认系统
>
> ❷ 默认系统，从 0 开始计数，每个*title*项为一个系统
>
> ❸ 标题，title 和分隔符后的内容为 Grub 菜单中显示的条目
>
> ❹ 引导分区
>
> ❺ 表示当前系统的 / 目录挂载到第一块硬盘的第三个分区 (hd0,2) ，内核根据该分区中的 `/etc/fstab` 文件来挂载文件系统
>
> ❻ 以只读模式挂载
>
> ❼ 显示启动画面
>
> ❽ 启动屏幕分辨率
>
> ❾ 设置活动分区
>
> ❿ 链式引导

Grub 安装

将 Grub 安装到系统中

```
grub-install  --no-floppy --root-directory=/boot /dev/sdM
```

> ❶ 不使用软盘
>
> ❷ 文件安装目录，通常不需要指定
>
> ❸ 目标磁盘

- 这一步只是拷贝 Grub 的文件。如果是修复系统，通常情况下它们已经存在，直接进行下一步就可以

将 Grub 安装到 MBR

`sudo grub` 进入 Grub 交互模式：

```
grub> find /boot/grub/stage2
 (hd0,0)

grub> root (hd0,0)
 Filesystem type is xfs, partition type 0x83

grub> setup (hd0)
 Checking if "/boot/grub/stage1" exists... yes
 Checking if "/boot/grub/stage2" exists... yes
 Checking if "/boot/grub/xfs_stage1_5" exists... yes
 Running "embed /boot/grub/xfs_stage1_5 (hd0)"...  19 sectors are embedded.
succeeded
 Running "install /boot/grub/stage1 (hd0) (hd0)1+19 p (hd0,0)/boot/grub/stage2
/boot/grub/menu.lst"... succeeded
Done.

grub> quit
```

> ❶ 搜索 stage2 文件所在的分区[39]，通常就是 Grub 根分区
>
> ❷ 返回 Grub 根分区名称
>
> ❸ 将这个分区设为 Grub 根分区
>
> ❹ 返回 Grub 根分区的文件系统信息
>
> ❺ 将 Grub 安装到 MBR。注意是(hd0)而不是(hd0,0)
>
> ❻ 检查需要的文件是否存在，因为用了 XFS 文件系统，所以需要 `xfs_stage1_5`
>
> ❼ 将 `xfs_stage1_5` 嵌入到 MBR，不然 Grub 不能够读取 XFS 分区中的 stage2
>
> ❽ 将 `stage1` 安装到 MBR，并指向 `stage2` 和 `menu.lst`
>
> ❾ 退出

- 可以使用 `TAB` 补全，或者按两次 `TAB` 列出提示

Grub 使用

在 Grub 启动菜单中，您可以选择您要的选项，按下 `d` 删除该项；或者按下 `e` 键，进入到编辑模式

修改您的启动参数，完成后回车

按 `b` 键，Grub 将以您修改后的参数引导系统。

在 Grub 启动菜单中，按下 `c` 进入命令模式，操作与交互模式基本相同；只不过最后一个命令是**boot**，而不是**quit**

第 23 章 服务器

目录

LAMP

`LAMP` 是一个缩写，它包括:Linux 操作系统，Apache 网络服务器，MySQL 数据库，Perl、PHP 或者 Python 编程语言。

毫不夸张的说，`LAMP` 是开源世界的“皇家海军”[42]，正是凭借 LAMP 的力量，开源应用得以在服务器市场称雄

web 服务器(Apache) 是 LAMP 的核心，几乎所有远程访问都要通过 web 服务器进行。除了 Apache ，web 服务器还有一些其它的开源实现，如：Lighttpd、Tomcat、Zope 等

Perl、PHP、Python 是服务器端脚本语言，web 服务器以扩展的形式支持这些语言作为后端处理器，以获得强大的可编程能力

数据库服务器(MySQL)提供数据存储、检索支持。除了 MySQL ，还有 PostgreSQL 等开源实现

理论上可以直接通过网络访问数据库服务器，不过出于安全需要，多数 LAMP 应用中，数据库服务器只能由 web 服务器本地访问。

也就是说，访问 web 服务器，产生了数据处理请求，web 服务器把数据请求发送到数据库服务器，然后将数据库服务器返回的处理结果发送给访问者

数据库服务器接受和返回数据处理请求，要通过 SQL 查询语言。web 服务器本身不支持 SQL，而服务器端脚本语言可以内嵌 SQL 语句，web 服务器通过脚本语言处理访问者对数据库服务器的请求

虚拟主机

一台机器上可能有多个网络接口：每一块网卡会分配一个 IP地址；类似 `127.0.0.1` 这样的回环地址则指向本地机器

网络接口就像港口，有很多泊位，称为端口。可以使用不同的协议连接(如 http、ftp、ssh 等)

服务器工作时，会监听某一地址的固定端口，称为"绑定"。如果该端口中传来请求，则进行响应

很多服务器可以针对不同的网络接口设置不同的响应规则，例如：从内网访问接口 `192.168.1.2` ，使用 `/home/user` 作为根目录；从公网访问接口 `211.148.131.7` 使用 `/var/www` 作为根目录

这就是虚拟主机

> [42] 或许 Emacs 工具链可以算是"太平洋舰队"

Lighttpd

`Lighttpd` 是一个新兴的、轻量级的 web 服务器，它开始越来越多的应用在一些重要场合，如：YouTobe、Sourceforge、豆瓣……

Lighttpd 以安全、快速和内存消耗低著称，还专门为大型分布式连接环境做了优化，支持 FastCGI, CGI, Auth, 输出压缩(output compress), URL重写, Alias 等重要功能。

Lighttpd 已经进入大多数发行版的软件仓库，安装方式见表 16.1 "包管理系统"

安装完成后，用启动脚本启动：`/etc/init.d/lighttpd start` ，见"手动控制服务"一节

`/etc/lighttpd/lighttpd.conf` 为 Lighttpd 服务器的配置文件[43]：

```
## 网站根目录 映射在机器上的物理路径
server.document-root        = "/home/lighttpd/html/"

## 如果网站目录中出现以下文件名，不用指定文件名便可直接访问
index-file.names            = ( "index.php", "index.html",
                                "index.htm", "default.htm" )

## Lighttpd 进程的归属用户
server.username             = "nobody"

## Lighttpd 进程的归属群组
server.groupname            = "nobody"

## 绑定到端口 默认为 80
#server.port                = 81

## 绑定到地址 默认为 所有
#server.bind                = "127.0.0.1"

## 访问日志 路径
accesslog.filename          = "/var/log/lighttpd/access.log"

## 错误日志 路径
server.errorlog             = "/var/log/lighttpd/error.log"

## 禁止访问以下文件
url.access-deny             = ( "~", ".inc" )

## 与目录列表相关的设置
#dir-listing.activate       = "enable"
#dir-listing.encoding       = "utf8"
#dir-listing.show-readme       = "enable"
```

配置文件中的 `server.modules` 字段决定Lighttpd使用哪些扩展模块：

```
server.modules = ("mod_access","mod_fastcgi","mod_accesslog" )
```

- Lighttpd 通过 `mod_fastcgi` 模块支持 PHP
- `mod_accesslog` 模块为访问纪录

其实在 `/etc/lighttpd/lighttpd.conf` 文件中，这部分内容写在多行，方便用 `#` 作注释，禁用不需要的模块

```
server.modules              = (
## 基础模块
                               "mod_access",
## 访问纪录
                               "mod_accesslog" )
## fastcgi 支持
                               "mod_fastcgi",
## cgi 支持
#                                "mod_cgi",
## 路径绑定
#                                "mod_alias",
##  代理 (转发页面)
#                                "mod_proxy",
## 虚拟主机
#                                "mod_evhost",
## 输出压缩
#                                "mod_compress",
## 网址重写
#                                "mod_rewrite",
## 用户认证
#                                "mod_auth",
#                                "mod_redirect",
#                                "mod_cml",
#                                "mod_trigger_b4_dl",
#                                "mod_status",
#                                "mod_setenv",
#                                "mod_simple_vhost",
#                                "mod_userdir",
#                                "mod_ssi",
#                                "mod_usertrack",
#                                "mod_expire",
#                                "mod_secdownload",
#                                "mod_rrdtool",
```

fastcgi 配置

在配置文件的 `server.modules` 字段中启用 `mod_fastcgi` 模块，然后检查以下内容：

```
### fastcgi 脚本扩展名
static-file.exclude-extensions = ( ".php", ".pl", ".fcgi" )

### fastcgi 服务器设置
fastcgi.server             = ( ".php" =>
                               ( "localhost" =>
                                 (
# TCP/IP 接口 (“套接字”)
                                   "socket" => "/tmp/php-fastcgi.socket",
# PHP cgi 模式的可执行文件(PHP 有 cli 和 cgi 两种模式)
                                   "bin-path" => "/usr/bin/php-cgi"
                                 )
                               )
                            )
```

上面例子的第二部分，使用 Lighttpd 转发规则。大意为： `.php` 文件按以下方式处理 => 从 `localhost` (本地)，发送到 `/tmp/php-fastcgi.socket` 接口，使用 `/usr/bin/php-cgi` 处理。写成一行比较直观：

```
    fastcgi.server  = ( ".php" => ( "localhost" => ( "socket" => "/tmp/php-fastcgi.socket
```

如果想要 fastcgi 和 PHP 协同工作，还需要对 PHP 作一些设置，见“PHP&MySQL”一节

proxy

该模块可以将文件转发到其它服务器进行处理，例如将 `.jsp` 文件转发到Tomcat服务器

```
### 首先启用 mod_proxy 模块
# += 表示在原来设置上增加
servers.modules +=( "mod_proxy")

### 设置 proxy 服务器转发规则
proxy.server               = ( ".jsp" =>
                                 ( "localhost" =>
                                   (
# 将 .jsp 文件发送到 地址“127.0.0.1”的“8080”端口(也就是本机的 Tomcat 服务器)
                                     "host" => "127.0.0.1",
                                     "port" => 8080
                                   )
                                 )
                               )
```

CGI

Lighttpd 可以支持 cgi

```
### 启用 mod_cgi 模块
server.modules              += ("mod_cgi")

### 设置 cgi 解释器
cgi.assign                 = ( ".pl"  => "/usr/bin/perl",
                               ".cgi" => "/usr/bin/perl",
                               ".py"  => "/usr/bin/python" )
```

路径绑定

将一个路径，映射到网站目录中

```
## 启用 mod_alias 模块
servers.modules +=( "mod_alias")

##  将 /home/lighttpd/html/man 映射到 http://host/docs
alias.url += ( "/docs" => "/home/lighttpd/html/man" )
```

虚拟主机

Lighttpd 可以建立多个虚拟主机，绑定在不同的网络接口

```
### 启用 mod_evhost 模块
servers.modules +=( "mod_evhost")

### 虚拟主机绑定的网络接口
$HTTP["host"] == "192.168.1.2"
{
### 虚拟主机可以使用独立的选项
dir-listing.activate       = "enable"
dir-listing.encoding       = "utf8"
dir-listing.show-readme       = "enable"
### 虚拟主机根目录
server.document-root = "/home/user/html"
### 虚拟主机路径绑定
alias.url = ( "/download/" => "/home/user/downloads/" )
alias.url += ( "/pictures/" => "/home/user/pictures/" )
}
```

> [43] 查看 `/etc/init.d/lighttpd` 文件，可以看到类似字句：
>
> `/usr/sbin/lighttpd -D -f /etc/lighttpd/lighttpd.conf`

- f 选项指定配置文件

PHP&MySQL

PHP 的配置文件为 `/etc/php/php.ini` ，如果与 Lighttpd 配合使用，请检查下面语句

```
cgi.fix_pathinfo=1
```

在 web 服务器根目录下新建一个 `index.php` 文件，内容如下：

```
<?php phpinfo(); ?>
```

访问这个页面，查看本服务器的 PHP 信息

与 Lighttpd 和 PHP 一样，MySQL 也进入了大多数发行版的软件仓库，使用包管系统安装，参见表 16.1 “包管理系统”

MySQL 安装后，可能需要重设密码：

```
## 停止 MySQL 服务器
/etc/init.d/mysqld stop
## 使用单用户维护模式运行
mysqld_safe --user=mysql --skip-grant-tables --skip-networking &
## 使用 root 身份，进入名为 `mysql` 的数据库
mysql -u root mysql
## 更新 表`user` 中，`USER`项 值为“root”的行，
## 设定这一行 `Password`项 的值，PASSWORD()函数用来给密码加密
mysql> UPDATE user SET Password=PASSWORD('这里设置密码') where USER='root';
mysql> FLUSH PRIVILEGES;
mysql> quit
## 重启 MySQL 服务器
/etc/init.d/mysqld restart

## 测试 MySQL 是否运行
$ mysqladmin -uroot -pmypasswd ping
mysqld is alive
## 测试 MySQL 密码
$ mysql -uroot -p
Enter password:
## 如果密码正确，会输出以下内容
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 6
Server version: 5.0.44-log Gentoo Linux mysql-5.0.44
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql>
```

PureFTPD

相对于ProFTPD、WuFTPD等老牌 ftp 服务器，PureFTPD、vsftpd这些轻量级 ftp 服务器更加实用

PureFTPD 的配置文件通常为 `/etc/pure-ftpd.conf` [44]

```
## 绑定的地址和端口(默认为所有 IP地址 的 21端口)
# Bind                        127.0.0.1,21

## 将所有用户限制在主目录中 (不能跳出)
ChrootEveryone                yes

## 如果前一个设置为 no，下面组的成员(GID)可以跳出主目录；其他用户仍然受限
## 如果想取消所有限制，注释掉 ChrootEveryone 和 TrustedGID
# TrustedGID                      100

## 如果用户主目录不存在，自动创建
# CreateHomeDir                 yes

## 新建目录及文件的权限掩码(禁止的权限)。文件掩码:目录掩码
Umask                         133:022

## 被动模式响应的端口范围
# PassivePortRange            30000 50000

## 强制一个 IP地址 使用被动模式
# ForcePassiveIP                  192.168.0.1

## 仅允许认证用户进行 FXP 传输 (服务器=>服务器)
AllowUserFXP                  yes

## 允许匿名 FXP 传输
AllowAnonymousFXP             no

## 兼容 ie 等山寨 ftp 客户端
BrokenClientsCompatibility  no
```

```
## 服务器允许的最大连接数
MaxClientsNumber            50

## 同一 IP 允许的最大连接数
MaxClientsPerIP             8

## 匿名用户最大带宽（KB/s）
# AnonymousBandwidth            8

## 所有用户最大带宽（KB/s），包括匿名用户
# UserBandwidth             8

# 匿名用户的上传/下载的比率
# AnonymousRatio                1 10

# 所有用户的上传/下载的比率
# UserRatio                 1 10

## 客户端的最大空闲时间（缺省15分钟，无动作将会被踢出）
MaxIdleTime                 15

## 不允许认证用户 (仅作为一个公共的匿名 FTP)
AnonymousOnly               no

## 不允许匿名连接，仅允许认证用户使用 (设置为 no 允许匿名连接)
NoAnonymous                 no

## 受信地址允许认证用户，其它地址只能匿名连接
#TrustedIP                  10.1.1.1

## 禁止匿名上传( no = 允许上传)
AnonymousCantUpload         no

## 是否允许匿名用户创建目录
AnonymousCanCreateDirs      no

## 'ls' 命令的递归限制。(文件的最大数目 最大子目录深度)
LimitRecursion              2000 8

## 保留所有文件(禁止删除)
# KeepAllFiles                yes

## 如果上传的文件已经存在，自动重命名
AutoRename                  no

## 禁止重命名
# NoRename               yes

## 禁止更改文件权限
# NoChmod                     yes

## 禁止读取隐藏文件（如 .history, .ssh 等)
ProhibitDotFilesRead        no

## 禁止下载所有者为 "ftp" 的文件 (匿名用户上传后未被本地管理员验证的文件)
AntiWarez                   yes

## 指定文件内容作为欢迎信息
# FortunesFile              /usr/share/fortune/zippy

## 启用磁盘限额。第一个数字为最大文件数，第二个数字为存储空间大小(单位：Mb)
## 1000:10 限制每一个用户只能使用 1000 个文件，共 10Mb
# Quota                       1000:10

## 最大可用空间，保证日志文件不会被覆盖 (默认为 99%)
MaxDiskUsage               99
```

PureFTPD 允许同时使用多种用户认证方式。以下为关于用户认证的配置：

```
## LDAP 配置文件
# LDAPConfigFile                /etc/pureftpd-ldap.conf

## MySQL 配置文件
# MySQLConfigFile               /etc/pureftpd-mysql.conf

## Postgres 配置文件
# PGSQLConfigFile               /etc/pureftpd-pgsql.conf

## PureDB 用户数据库
# PureDB                        /etc/pureftpd.pdb

## pure-authd 套接路径
# ExtAuth                       /var/run/ftpd.sock

## 启用 PAM 认证
# PAMAuthentication             yes

# 启用 Unix 系统认证 (/etc/passwd)
# UnixAuthentication            yes
```

PureFTPD 可以使用 syslog 生成日志，以下为关于日志的配置：

```
## Syslog 日志 。默认为 "ftp"， "none" 禁用日志
SyslogFacility              ftp

## 在日志文件中不解析主机名
DontResolve                 yes

## 在日志中添加 PID
# LogPID                     yes

## 使用 Apache 格式创建额外日志
# AltLog                     clf:/var/log/pureftpd.log

## 使用优化格式创建额外日志
# AltLog                     stats:/var/log/pureftpd.log

## 使用 W3C 格式创建额外日志
# AltLog                     w3c:/var/log/pureftpd.log
```

> [44] 查看 `/etc/init.d/pure-ftpd` 文件，可以看到类似字句：
>
> `/usr/sbin/pure-config.pl /etc/pure-ftpd.conf`

启用配置文件

第 24 章 Vim 编辑器

目录

加密

版本

VIM 简介

我们使用的大多数编辑器，都可以直接在编辑区输入字符，并且能够通过一些快捷键来完成一些控制功能，比如使用方向键移动光标，使用 `BackSpack` 或者 `Delete` 键删除文字，使用 `PgUp` 和 `PgDn` 翻页，使用 `Home` 和 `End` 来定位行首和行末……

而Vim是一个带模式的编辑器，同样的按键，在不同模式下，具有不同的功能定义。例如 `h j k l` 在 编辑模式 下输入相应的字符，在 普通模式 下却相当于方向键的作用。

由于需要切换模式，Vim 的使用起来略显繁琐。不过优点也显而易见：您只要把手安安稳稳的放在打字区就可以了，而不需要使用诸如方向键、排版键、小键盘等需要挪开双手的键位，从而提高了您的效率和专注程度。事实上，Vim 的前身 Vi 诞生的时候，键盘上还没有方向键、排版键和小键盘

命令

使用 Vim 编辑文件:

```
vi [文件名]
vim [文件名]
```

教学模式:

```
vimtutor [语言]
```

vim 教程，相当于使用Vim编辑器以只读模式打开教程文件。您无论对这个文件作了什么，都会在退出后恢复原来的样貌。与只读模式的区别在于，它不会没有眼色的提醒您，现在的状态为只读模式。您可以使用它作一些练习

您可以指定教程文件的语言，如果使用本地语言(**ZH_cn**)出现乱码，您可以尝试使用英语

```
vimtutor en
```

使用 Vim 比较文件区别

```
vimdiff [文件1] [文件2] [其它文件]……
```

配置文件

Vim 的全局配置文件为 `/etc/vim/vimrc` ，用户配置文件为 `~/.vimrc` ， `"` 起始的行为注释行。我们提供的配置项，您直接加入配置文件就可以了

您可以先对 Vim 进行一些简单的配置：

例 **24.1. Vim** 配置 `/etc/vim/vimrc`

```
"运行在非兼容(VI)模式下(命令模式下 TAB 补全)
set nocompatible

"设定文件编码
set fileencodings=utf-8,ucs-bom,gb18030,gbk,gb2312,cp936

"开启语法加亮
syntax on
"配色风格
colorscheme pablo

"简洁启动模式
set shortmess=atI
"设定 GUI 选项
"set guioptions=gmrLtT  m:菜单 T:工具栏
set guioptions=gmrLt
"命令行高度
set cmdheight=1
"设定行距 ( GUI 界面中生效 )
set linespace=4
"显示标尺
set ruler

"自动折行
"set nowrap
set wrap
"按完整单词折行
set nolinebreak
"set linebreak
"行宽（输入时自动插入换行符）
"set textwidth=80
set textwidth=0

"允许在 虚空间 内操作 （虚空间:不包含任何文本的空间。如换行符之后）
"set virtualedit=all
"禁止在 虚空间 内操作
set virtualedit=

"设定 Tab 键缩进的空格数
set tabstop=4
"设定编辑器将多少空格视为一个缩进
set shiftwidth=4
"将缩进转换为空格
"set expandtab
"设定自动缩进(新行与前一行缩进相同)
set autoindent
"set noautoindent
"插入模式下，“←”如何删除光标前的字符：行首空白、换行符、插入点之前的字符
set backspace=indent,eol,start

"命令行历史纪录
set history=500

"禁用增量搜索
set incsearch
"set noincsearch
"搜索时忽略大小写
set ignorecase
"set noignorecase
"高亮显示搜索结果
set hlsearch

"设定折叠方式
"set foldmethod=manual

"以下字符将被视为单词的一部分 (ASCII)：
"set iskeyword+=33-47,58-64,91-96,123-128
```

模式行

也可以通过“模式行”将配置选项嵌入在普通文件中。打开此文件时，优先使用模式行中的配置：

```
# **vim:** filetype=sh foldmethod=marker autoindent expandtab shiftwidth=4
模式行可以写在任意位置
但为了不影响文件功能，模式行通常写在注释中
模式行以 **vim:** 起始，前面至少要有一个空白字符（空格、换行符等）
模式行中的设置项以空格分隔
……
```

这种形式的缺点在于，不能在设置项后添加其它文本。如果有相关需要，可以使用另一种模式行：

```
<!-- **vim:set** filetype=html**:** 这里可以任意添加文本 -->
```

模式介绍

Vim 常见的模式有： 普通模式 、 插入模式 、 命令模式 ，另外我们也会经常用到 可视模式 。

Vim 启动时进入 普通模式 ；或者在其它模式下，按下 `Esc` 键，便可以回到普通模式。

使用 **vimtutor en** 命令进入教程，现在是普通模式。随便按几下 `j` 、 `k` 、 `l` 、 `h` 键，您会发现光标的位置发生改变。

按下 `i` 键，编辑器底部出现了 -- 插入 -- 或者 -- insert -- ，您进入了插入模式。随便按几下 `j` 、 `k` 、 `l` 、 `h` ，您会发现相应的字符出现在编辑区，现在还可以通过方向键来移动光标。可能您觉得使用方向键移动光标不是什么问题，但是习惯了 Vim 后，您反而会认为方向键太麻烦了，简直不能容忍！好了，现在按下 `Esc` 键回到普通模式，我们又可以使用 `j` 、 `k` 、 `l` 、 `h` 来移动光标了。

在普通模式下，按下 `:` 键（也就是 Shift+; ），在编辑器底部出现了一个 : ，您进入了命令模式。在 `:` 后输入一个命令 `new` ，回车后，编辑器被分割为上下两栏。为了方便起见，我们在命令前加一个 **:** 来表示命令模式下输入的命令，像这样

```
:vnew
```

命令能够以一些规则简化，上面的命令也可以写为这种形式

```
:vne
```

现在您的编辑区一定弄的四分五裂，您可以使用命令“**:quit**”来关闭当前栏，直接用简写就可以了

```
:q
```

这个命令是退出编辑器，如果编辑区被分成多栏，则是退出当前栏。

执行完一个命令（按下回车后），编辑器会自动回到普通模式。如果想不执行当前命令，直接回到普通模式，您可以按下 Esc键。

按下 `v` 键，您进入了可视模式，可以使用 `j` 、 `k` 、 `l` 、 `h` 移动光标，高亮选取文本。

事实上，可视模式相当于高亮选取文本后的普通模式。

可视模式具有多种模式，以行为单位进行选取的可视行模式，使用 `V` 键进入（也就是 Shift+v ）；和以块为单位进行选取的可视块模式，使用 `Ctrl+v` 键进入。

模式切换

好了，现在我们总结一下模式间切换的方法

其它模式	普通模式	Esc	
普通模式	插入模式	i	在光标前插入
I	在行首插入		
a	在光标后插入		
A	在行末插入		
o	在当前行之下新建行		
O	在当前行之上新建行		
r	替换当前字符		
R	从当前字符开始替换		
普通模式	命令模式	:	
普通模式	可视模式	v	可视模式
V	可视行模式		
Ctrl+v	可视块模式		

移动

在普通模式中，您可以使用以下方式移动光标

k(上)	
h(左)	l(右)
j(下)	

您可以使用其它更有效率的方式移动光标

表 **24.1.**

	向前	向后
单词	w	b
单词，包括特殊符号	W	B
单词词尾	e	
单词词尾，包括特殊符号	E	
行	0	$
行首文字（不包括空格）	^	
页	H	L
页面中部	M	

在其它模式中，您可以使用方向键移动光标，不过那样比较麻烦，您可以在配置文件中绑定插入模式下的功能键

```
noremap! <M-j> <Down>
noremap! <M-k> <Up>
noremap! <M-h> <left>
noremap! <M-l> <Right>
………………
作用范围 键位 功能
```

- 其中，**map!**绑定的键盘映射，作用于所有模式；**inoremap!**绑定的映射，仅作用于插入模式。

数字参数

您也可以使用数字参数(普通模式)，来重复执行。例如

```
100j
```

- 执行 100次 `j` 键，向下 100行

浏览

表 **24.2.**

	向前	向后
整页	Ctrl+f	Ctrl+b
半页	Ctrl+d	Ctrl+u
文件	G	gg
按行号转到相应行	行号 G	
按百分比转到相应的行	1~100%	
按行卷动	Ctrl+e	Ctrl+y
将光标所在行调整至页面中间	zz	
统计字数	g Ctrl+g	
显示位置信息	Ctrl+g	
刷新屏幕	Ctrl+l	

> 提示：`gg` 定位到文件首行， `V` 进入可视行模式， `G` 定位到文件末行，实现类似“全选”的功能。依次按下 `g` `g` `V` (Shift+v) `G` (Shift+g)

标记

您可以在当前光标处作一个标记，以便快速返回

```
m标记名称      定义标记。标记名称为一个字符
`标记名称      返回标记

mx            将当前光标处定义为标记 x
`x            返回标记 x

:marks        查看标记列表
```

编辑

	复制	剪切
字符	y❶	x
行	yy	dd

> ❶ 需先在可视模式中选取

p	在光标后粘贴
P	在光标前粘贴
u	撤消
Ctrl+r	重做
Ctrl+y	逐字克隆上一行内容
Ctrl+e	逐字克隆下一行内容

搜索和替换

按下 `/` 键，编辑器底部会出现 `/` 符号，接着输入字符串，便可以进行搜索

/	向下搜索	?	向上搜索
n	搜索下一个	N	搜索上一个

:s/源字符串/目标字符串	将源字符串替换为目标字符串
:s/源字符串/目标字符串/g	替换当前行中所有符合条件的字符串
:行号1,行号2s/源字符串/目标字符串/g	在指定行中进行替换
:%s/源字符串/目标字符串/g	全文替换

正则表达式

见第 26 章 正则表达式

寄存器操作

Vim 可以将不同字段剪切或复制到不同寄存器中，您可以从不同寄存器中取出内容后粘贴

```
"寄存器名称
```

- 按下 `"` 键和另一个字符键，便可以定义一个寄存器。例如：`"a` `"1`

定义寄存器后直接进行操作

```
"ayy    将当前行复制到寄存器 a 中
"ap     将寄存器 a 中的内容粘贴到光标之后
```

- **:registers** 查看所有寄存器的内容
- 通常情况下，寄存器 **+** (先按"，再按 Shift+=)对应 X 下的剪贴板。您在其它程序中复制的内容，可以使用 `"+p` 粘贴到 Vim中；您在 Vim 中，可以使用 `"+y` 将内容复制到剪贴板，再粘贴到其它程序中
- 没有指定寄存器时，Vim 使用“无名寄存器”存储内容

宏

您可以将一系列的操作录制为一个宏，然后执行它

```
q宏名称        开始录制宏。宏名称为一个字符
q              录制中按下“q”键，结束录制

@宏名称        执行宏
```

- 可以使用 **:registers**（寄存器列表） 命令查看已记录的宏

插入模式下的快捷键

Ctrl+r寄存器名称	插入指定寄存器内容
Ctrl+k(2个字符)	输入二合字符
Ctrl+v数字	通过数字编码输入字符
Ctrl+v键位	输入键位的名称

键绑定、缩写

前面我们已经向您介绍了键绑定

```
map! <M-j> <Down>
```

尖括号及其中的内容，为 Vim 配置文件的约定，分别描述了按键和功能，表示将功能编写到按键上。如果绑定的只是普通字符，例如：

```
map! xxx XXXXX
```

表示将 XXXXX 绑定到 xxx 上。当您键入 xxx 时，编辑器会自动替换为 XXXXX

如果您只是想将字符串绑定为缩写，方便输入，我们建议您使用 **iabbrev** 来绑定。例如：

```
iabbrev g google
```

在插入模式下键入 `g` ，编辑器会自动替换为 google 。您可以将 **iabbrev** 命令缩写为 **iab** ，例如：

```
iab g google
```

以上命令，您可以直接在命令模式下输入，临时启用。也可以写入配置文件，永久启用。

单词补全

```
<Ctrl+n> 下一个匹配项
<Ctrl+p> 上一个匹配项
```

您可以在配置文件中定义补全的方式

```
"自动补全方式：(使用逗号分隔)
set complete=k,.

"   .     当前文件
"   b     已被装缓冲区,但是没有在窗口内的文件
"   d     在当前的文件中定义和由 #include 包含进来的文件
"   i     由 #include 包含进来的文件
"   k     由 dictionary 选项定义的文件
"   kfile    名为{file}的文件
"   t     标记(tags)文件
"   u     没有载入的缓冲区
"   w     在其他窗口中的文件

"设定自动补全字典 :
set dictionary=path
```

命令模式

前面介绍了普通模式和插入模式。我们发现，普通模式主要用来浏览和修改文本内容，而插入模式则用来向文本中添加内容。

而命令模式则多用于操作文本文件（而不是操作文本文件的内容），例如保存文件；或者用来更改编辑器本身的状态，例如设定多栏窗口、标签或者退出编辑器……

w(rote) 将更改写入文件

```
:w
```

q(uit) 退出编辑器

```
:q
```

某些情况下，编辑器会阻止命令的执行。例如您修改了文件，而没有保存，那么您使用 :q 命令退出时，编辑器就不会执行这条命令，而是提醒您保存文件。

这个时候，您可以在命令末尾追加！来强制执行命令

```
:命令!
```

例如 `:q!` ，即便您没有保存已修改的文件，使用此命令，编辑器也会放弃修改而强行退出

以 `!` 引导一个 Shell 命令，则可以从 Vim 临时切换到 Shell 中，执行一个 Shell

```
:!命令
```

例如 `:!ls`

命令模式还可以用来调节一些选项，例如

```
:set linespace=6
```

- 使用 Tab 键补全命令
- 命令可以直接写入配置文件
- 在选项后加上一个 `&` 使用默认值，例如： `:set linespace&`

折叠

zf数字	创建折叠，数字参数为折叠行数
zo	打开折叠
zc	关闭折叠
zd	删除折叠
:set foldcolumn=4	显示折叠树
:mkview	保存折叠
:loadview	读取折叠
zr	打开同一层级所有折叠
zm	关闭同一层级所有折叠
zO	打开某一行的所有层级折叠
zC	关闭某一行的所有层级折叠
zR	打开所有折叠
zM	关闭所有折叠
:set foldclose=all	光标离开折叠自动关闭
:set foldopen=all	光标遇到折叠自动打开

您可以在配置文件中定义折叠的方式

```
"设定折叠方式
set foldmethod=manual

" manual 手动折叠
" indent 按缩进折叠
" marker 按标记折叠
" syntax 按语法折叠
" expr   按表达式折叠
```

按标记折叠的例子：

```
#起始标记 **{{{** 将标记放到注释里，不会影响文件的功能

起始标记和结束标记之间的内容折叠
显示为起始标记所在行
#结束标记 **}}}**
###### 将折叠方式写到模式行里 #####
# vim: foldmethod=marker
```

多栏窗口

您可以使用以下命令，将当前窗口水平分为两栏

```
:new
```

新建一栏空白窗口，将当前文件分两栏显示

```
:split
```

同理，您可以使用下列命令，将当前窗口垂直分为两栏

```
:vnew
:vsplit
```

先按下 `ctrl+w` 键，再按下方向键 `j` 、 `k` 、 `l` 、 `h` ，您可以切换到其它栏；在当前栏中使用 `:q` 命令，可以退出当前栏，也可以使用其它命令，对当前栏作出修改

如果您希望当前命令在所有栏中生效，您可以在命令的末尾追加 **all**

```
:命令all
```

例如： `:qall`

如果您希望这条命令强制执行，那么 `!` 位于命令的最末，例如： `:qall!` 强行退出所有栏窗口

标签页

Vim 在7以后的版本，开始支持标签页的功能

```
:tabnew      新建一个标签
:tabnext     转到下一个标签
:tabprevious   转到上一个标签
```

您也可以使用鼠标点击标签进行切换

文件管理

使用 Vim 编辑目录时，会进入文件管理模式：

```
vim .
```

文件管理模式中可以实现一些简单的文件管理功能：

enter	打开文件或文件夹
Ctrl+o	后退
o	新开一栏进行编辑
O	在前一次打开的栏中编辑
p	预览
i	显示/隐藏文件大小、日期
s	切换文件排序方式
r	切换顺序、逆序
-	回到上一级目录
c	将 Vim 执行目录设定为当前目录
R	重命名
D	删除
x	运行

- **:cd** 设定所有窗口执行目录 ； **:lcd** 设定当前栏窗口执行目录

加密

使用 **:X** 命令为文件设定一个密码

```
:X
Enter encryption key: ******
Enter same key again: ******
```

- 如果要取消加密，可以设置密码为空

可以禁用交换文件，以免泄密

```
vim -x -n file.txt
```

如果你已在经编辑这个文件了，那么交换文件 `swapfile` 可以用下面的命令禁止:

```
:setlocal noswapfile
```

由于没了交换文件，文件复原就不可能了。为了避免失去编辑的成果，要比平时更勤快地存盘你的文件

现在你可以像平时一样编辑这个文件并把你所有的秘密放进去。当你编完文件要退出 Vim 时，这个文件就被加密存盘了

当你下次用 Vim 编辑这个文件时，它就会询问你密码

如果你试图用另一个程序来阅读这个文件，你将读到一堆垃圾。如果你用 Vim 来编辑这个文件，但输入了错误的密码，你也只能得到垃圾。Vim 并不具备检验密码正确性的机制 (这一点使得破译密码更为困难)

版本

在 Windows 平台下，请使用 gvim7ole 版本

第 25 章 Emacs 入门

目录

基础知识

引起我们痛苦的东西我们就会去爱它，以使自己觉得这份痛苦是值得的

Emacs 是一个架构在编辑器上的集成环境，除了最基本的编辑功能，还可以完成文件管理、终端模拟、浏览网页、收发邮件、编译程序等工作。

Emacs 使用 Elisp 语言进行配置和扩展，它本身也可以作为 Elisp 解释器使用。

Emacs 的界面主要由三部分构成：信息栏、回显区(echo)、缓冲区(buffer)

缓冲区

缓冲区(buffer) 类似于常规编辑器的文字编辑区。Emacs 并不直接对文件进行修改，而是读取文件的内容并显示在缓冲区中，在收到保存的指令后才将修改写入文件。

缓冲区名称通常为它所读取文件的文件名。

信息栏

在缓冲区之下为状态栏，像这样的



简单说明一下：



> ❶ 点击拖动这里可以状态栏位置
>
> ❷ 文件编码 U 代表 UTF-8； c 代表 chinese-gbk
>
> ❸ 换行符类型 有 UNIX、DOS 和 MacOS 三种
>
> ❹ 文件状态 %代表只读 -代表可写 *代表未保存
>
> ❺ 当前工作路径
>
> ❻ 当前编辑的文件
>
> ❼ 光标在当前文件中的位置
>
> ❽ 光标所在的行、列
>
> ❾ 版本控制系统
>
> ❿ 主模式
>
> ⓫ 辅模式
>
> ⓬ 日期时间

标题栏也可以显示一部分信息，并且可以自由定义。

按键描述

Emacs 的功能键，通常为组合键。例如

表 **25.1. Emacs** 按键描述

Emacs 描述	实际热键	功能
C-f	Ctrl+f	光标前进一格
C-b	Ctrl+b	光标后退一格
C-d	Ctrl+d	删除一个字符
C-a	Ctrl+a	回到行首

Emacs 对按键的描述方式中， - 之前的一个字符为修饰键，表示按住该键，再按 - 后面的键。

例如： `C-a` 表示按住 Ctrl 再按 a 键。 Emacs 对其它一些特殊按键的描述

`C-`	按住 Ctrl键
`M-`	按住 Meta键。在 PC 上，Meta键 通常对应 Alt 键
`C-M-`	同时按住 Ctrl键 和 Meta键
`S-`	Shift键
`s-`	Linux 下对应 WIN 键
`RET`	回车键
`TAB`	Tab键
`ESC`	Esc键
`SPC`	空格键
`DEL`	退格键
`Delete`	删除键

> 注意：在 Emacs 中，`ESC` 等价于 `M-`，也就是说，按一次 `ESC` 键，与按住 `Meta` 键的作用是相同的。
>
> 类似 `C-M-r` 的按键，可以使用 `ESC C-r`(按一次 `ESC`，再按 `C-r`)这种更容易的方式

在后面的部分中，将统一使用 Emacs 对按键的描述方式。

Emacs 十分强大，上面的组合键，远不能涵盖 Emacs 的功能于万一。除基本的编辑功能键外，其它功能多使用 按键序列： 连续的按下多组快捷键

例如： `C-x C-c` 表示先按下 `C-x` ，再按下 `C-c` 。也就是 `Ctrl+x` 后，再 `Ctrl+c` (退出 Emacs)

接下来 `C-h t` ，进入 《Emacs 快捷指南》

回显区

`C-x h` （先按 Ctrl+x 再按 h) 后，您会发现状态栏和编辑器底部之间的区域出现 Mark set 字样。同时，整个缓冲区的内容都被选中。

它是一个迷你缓冲区(minibuffer)，叫作回显区(echo area)，提示您正在进行的操作，比如 Mark set(设定标记)

如果一个按键序列没有完成，却停止了输入。大约两秒后，回显区会显示已输入部分，以免您忘记。不要以为是 Emacs 反应迟钝

内部命令

`C-h k` 后，回显区提示

```
Describe key (or click or menu item):
```

接着 `C-x h` ，您会发现，缓冲区被水平分割为两个。另一个名为 *help* 缓冲区中显示的内容为

```
C-x h runs the command mark-whole-buffer
   which is an interactive compiled Lisp function in `simple.el'.
It is bound to C-x h, <menu-bar> <edit> <mark-whole-buffer>.
(mark-whole-buffer)

Put point at beginning and mark at end of buffer.
You probably should not use this function in Lisp programs;
it is usually a mistake for a Lisp function to use any subroutine
that uses or sets the mark.

[back]
```

第一行说明了 `C-x h` 运行的命令为 **mark-whole-buffer**

第二行说明了该命令由 `simple.el` 这个扩展提供，绑定到 `C-x h` 、 菜单栏-编辑-标记全部缓冲区 、 和命令 **mark-whole-buffer**

第三行介绍了这个命令的行为： 在文档末尾设置一个标记，并把光标点[45]移动到文档起始。

> 注意：*Emacs* 使用命令进行处理，快捷键只是一种发送命令的方法！

一般情况下，我们用不到这么详细的说明，而且英文看起来也比较吃力。您可以使用 `C-h c` 以简洁模式查看说明。 只在回显区显示键位和它执行的命令：

```
C-x h runs the command mark-whole-buffer
```

通常这就足够了。

如果您知道一个命令，而不知道它绑定到什么键上，您可以使用 `C-h w` ，也就是命令 **Where-is**

Emacs 命令行

由于 Emacs 太过强大，内部命令恒河沙数，根本不可能有同样数量的快捷键位来绑定它们！

对于没有绑定的命令，可以使用命令执行！

`M-x` (Alt+x) 开启命令行，回显区显示为 M-x ，然后输入 newline

这个命令默认绑定在回车键，所以它和回车键的作用一样为 换行

C-h w newline 结果是： newline is on RET

> 提示：命令行中，可以使用 `TAB` 补全，使用 `M-p` 上翻， `M-n` 下翻

在后面的部分中，统一使用 `M-x command` 来表示 内部命令 **command** ；内部命令以 "(command)" 的形式写到配置文件中

Emacs 终端

`M-x shell` 激活 Emacs 终端。可以在 Emacs 终端中使用外部命令。

需要注意的是，Emacs 终端是哑终端，某些类型的输出不能够正确显示。

在 Emacs 终端中使用 `exit` 命令退出。

`M-!` (Alt+Shift+1)临时执行一条外部命令，并输出在名为 *Shell Command Output* 的缓冲区中 **(M-x shell-command)**

`C-u M-!` (Ctrl+u Alt+Shift+1)临时执行一条外部命令，并输出到光标位置。

Emacs 文件管理器

`C-x d` 进入 Dired 列表模式

`C-x C-d` 获取文件列表（简洁）

`C-x C-f` 打开文件，输入路径为打开目录

详细介绍见“文件管理”一节

区块选择

很多时候，我们需要选中缓冲区中的某一部分内容。和大多数程序一样，您可以在被选择区块的起始点按下左键，移动鼠标，在结束点释放左键，这部分区块便被选中。

这种方式效率并不高，而且一些场合并没有鼠标支持，例如控制台或者远程登录。

事实上，Emacs 进行区块选择的方法，是设置一个标记，标记到光标点[45]之间的部分将被选中。

标记的位置为 **M-x set-mark-command** 时，光标点[45]所处的位置。

M-x set-mark-command 是设置标记的内部命令，默认绑定在 `C-SPC` 键上。

如果使用输入法，这个键位多半是切换输入法的快捷键。键盘指令会先被输入法拦截下来，而无法发送到 Emacs。

当然也可以使用 `M-@` 来设定标记。不过 `M-@` 原绑定为 **M-x mark-word** ，虽然差不太多，但有时并不好用；况且对于一个常用的命令来讲， `M-@` 键位的难度太高了

Emacs 的键位中，几乎没有默认绑定在 WIN 键上的命令，不妨利用一下

在 Emacs 的用户配置文件 `~/.emacs` 中添加如下内容：

```
;; WIN+Space 设置标记
(global-set-key (kbd "s-SPC") 'set-mark-command)
```

在某些类型的终端中，WIN键 不起作用，建议使用命令。或者绑定到 `C-t` [46]

```
;; （在注释里说明原命令和绑定，是一个良好的习惯）
;; C-t 设置标记
(global-set-key (kbd "C-t") 'set-mark-command)
```

> 重要：重启 Emacs，或者在 ~/.emacs 文件的缓冲区中执行命令 `M-x eval-buffer` ，便可以使配置文件立即生效

中止执行

如果想放弃一个命令，可以使用 `C-g` **(M-x keyboard-quit)**打断。

建议您使用快捷键 C-g ，因为在需要中止执行的情况下， M-x 通常是无法使用的

`ESC ESC ESC` **(M-x keyboard-escape-quit)**可以从一些交互命令中退出。

例如从 "询问替换 M-x query-replace" 中退出。

当 C-g 不能搞定，您可以尝试连按三次 ESC

> [45] 光标点假定光标为插入式（竖线），位置在覆盖式光标(方块)的左侧。
>
> 事实上，Emacs 中的相关判定以光标点为准！方块形光标只是为了减少视觉疲劳
>
> [46] 这是一个让人头痛不已的地方。因为无论绑到哪，似乎都不太方便：
>
> 使用 WIN键 倒是挺好，但是在字符界面下，WIN键 通常不起作用；同样，C-; 这样使用标点的组合键在字符界面下也不行;
>
> C-m、C-i 是两个不错的组合，但是 Emacs 认为 C-m 和 RET、C-i 和 TAB 是一个键，这样绑定，你的回车或者 TAB 就成了设置标记；
>
> 最后，C-t 这个键默认绑定的命令几乎没什么用，只是这个键位不是很好按，但这样也有好处——无论你用左手或者右手来按 t键，距离都差不多

基本配置

您已经知道了，Emacs 的配置文件为 `~/.emacs` 。配置文件中，以 ; 起始到行末的部分为注释。

让我们些简单配置一下：

例 25.1. emacs 配置 `~/.emacs`

```
;;=======================================
;;添加 Emacs 搜索目录 可以将自定的扩展放该目录
;;=======================================
;(add-to-list 'load-path "~/..emacs")
;如果有其它配置文件，使此命令读取
;(load "addon.el")

;;=======================================
;; 外观设置
;;=======================================

;;禁用工具栏
(tool-bar-mode nil)

;;禁用菜单栏，F10 开启关闭菜单
(menu-bar-mode nil)

;;禁用滚动栏，用鼠标滚轮代替
;;(scroll-bar-mode nil)

;;禁用启动画面
(setq inhibit-startup-message t)

;;=======================================
;; 键绑定
;;=======================================

;; C-t 设置标记 ;;
(global-set-key (kbd "C-t") 'set-mark-command)

;; C-x b => CRM bufer list
(global-set-key "\C-xb" 'electric-buffer-list)

;;---------- redo
(global-set-key ( kbd "C-.") 'redo)

;;=======================================
;;关闭当前缓冲区 Alt+4  ;; C-x 0
(global-set-key (kbd "M-4") 'delete-window)
;;关闭其它缓冲区 Alt+1  ;; C-x 1
(global-set-key (kbd "M-1") 'delete-other-windows)
;;水平分割缓冲区 Alt+2  ;; C-x 2
(global-set-key (kbd "M-2") 'split-window-vertically)
;;垂直分割缓冲区 Alt+3  ;; C-x 3
(global-set-key (kbd "M-3") 'split-window-horizontally)
;;切换到其它缓冲区 Alt+0 ;; C-x o
(global-set-key (kbd "M-0") 'other-window)

;;F10 显示/隐藏菜单栏 ;; M-x menu-bar-open
;;(global-set-key (kbd "F10") 'menu-bar-mode)

;;WIN+s 进入 Shell ;; M-x shell
;(global-set-key (kbd "s-s") 'shell)
;;(define-key ctl-x-map "\M-s" 'shell)

;;=======================================
;; 缓冲区
;;=======================================

;;设定行距
(setq default-line-spacing 0)
```

```
;;页宽
(setq default-fill-column 90)

;;缺省模式 text-mode
(setq default-major-mode 'text-mode)

;;设置删除纪录
(setq kill-ring-max 200)

;;以空行结束
(setq require-final-newline t)

;;语法加亮
(global-font-lock-mode t)

;;高亮显示区域选择
(transient-mark-mode t)

;;页面平滑滚动， scroll-margin 5 靠近屏幕边沿3行时开始滚动，可以很好的看到上下文。
(setq scroll-margin 5
      scroll-conservatively 10000)

;高亮显示成对括号，但不来回弹跳
(show-paren-mode t)
(setq show-paren-style 'parentheses)

;;鼠标指针规避光标
;(mouse-avoidance-mode 'animate)

;;粘贴于光标处，而不是鼠标指针处
(setq mouse-yank-at-point t)

;;========================================
;; 回显区
;;========================================

;;闪屏报警
(setq visible-bell t)

;;使用 y or n 提问
(fset 'yes-or-no-p 'y-or-n-p)

;;锁定行高
(setq resize-mini-windows nil)

;;递归 minibuffer
(setq enable-recursive-minibuffers t)

;; 当使用 M-x COMMAND 后，过 1 秒钟显示该 COMMAND 绑定的键。
;;(setq suggest-key-bindings 1) ;;

;;========================================
;; 状态栏
;;========================================

;;显示时间
(display-time)
;;时间格式
(setq display-time-24hr-format t)
(setq display-time-day-and-date t)
(setq display-time-interval 10)

;;显示列号
(setq column-number-mode t)

;;标题栏显示 %f 缓冲区完整路径 %p 页面百分数 %l 行号
(setq frame-title-format "%f")

;;========================================
;; 编辑器设定
;;========================================
```

```
;;不生成临时文件
;;(setq-default make-backup-files nil)

;;只渲染当前屏幕语法高亮，加快显示速度
(setq font-lock-maximum-decoration t)

;;将错误信息显示在回显区
;(condition-case err
;    (progn
;    (require 'xxx) )
;  (error
;   (message "Can't load xxx-mode %s" (cdr err))))

;;使用X剪贴板
(setq x-select-enable-clipboard t)
;;;;;;;; 使用空格缩进 ;;;;;;;;
;; indent-tabs-mode  t 使用 TAB 作格式化字符  nil 使用空格作格式化字符
(setq indent-tabs-mode nil)
(setq tab-always-indent nil)
(setq tab-width 4)

;;========================================
;; 颜色设置
;;========================================

;; 指针颜色
(set-cursor-color "black")
;; 鼠标颜色
(set-mouse-color "black")
;; 背景和字体颜色
(set-foreground-color "gainsboro")
(set-background-color "grey30")
(set-border-color "black")
;; 语法高亮显示，区域选择，二次选择 ;;前景和背景色
(set-face-foreground 'highlight "white")
(set-face-background 'highlight "blue")
(set-face-foreground 'region "cyan")
(set-face-background 'region "blue")
(set-face-foreground 'secondary-selection "skyblue")
(set-face-background 'secondary-selection "darkblue")

;;日历配色
;(setq calendar-load-hook
;'(lambda ()
;(set-face-foreground 'diary-face "skyblue")
;(set-face-background 'holiday-face "slate blue")
;(set-face-foreground 'holiday-face "white")))

;;========================================
;; 字体设置
;;========================================
(set-default-font "Monospace-10")
(if window-system
   (set-fontset-font (frame-parameter nil 'font)
      'unicode '("Microsoft YaHei" . "unicode-bmp"))
)

;;========================================
;; 必备扩展
;;========================================

;;---------- color-theme
;(require 'color-theme)
;(color-theme-gray30)

;;========== ibuffer
(require 'ibuffer)
(global-set-key ( kbd "C-x C-b ")' ibuffer)

;;========== outline
(setq outline-minor-mode-prefix [(control o)])
```

```
;;---------- Docbook
;(require 'docbook-xml-mode)

;(add-hook 'docbook-xml-mode-hook
;      (function (lambda ()
;                   (setq outline-regexp "<!\\-\\-\\*+")
;               (outline-minor-mode)
;               (hide-body)
;                 )))

;;开启服务器模式
;(server-start)

;;org-mode
(setq org-hide-leading-stars t)
 (define-key global-map "\C-ca" 'org-agenda)
 (setq org-log-done 'time)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;; F5 运行当前文件 ;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;(defun run-current-file ()
;  "Execute or compile the current file.
;For example, if the current buffer is the file x.pl,
;then it'll call “perl x.pl” in a shell.
;The file can be php, perl, python, bash, java.
;File suffix is used to determine what program to run."
;(interactive)
;  (let (ext-map file-name file-ext prog-name cmd-str)
;; get the file name
;; get the program name
;; run it
;    (setq ext-map
;          '(
;            ("php" . "php")
;            ("pl" . "perl")
;            ("py" . "python")
;            ("sh" . "bash")
;            ("java" . "javac")
;            )
;          )
;    (setq file-name (buffer-file-name))
;    (setq file-ext (file-name-extension file-name))
;    (setq prog-name (cdr (assoc file-ext ext-map)))
;    (setq cmd-str (concat prog-name " " file-name))
;;    (compile cmd-str)))
;    (shell-command cmd-str)))
;
;(global-set-key (kbd "<f5>") 'run-current-file)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;; 以下为实现 redo 的代码 ;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(provide 'redo)

(defvar redo-version "1.02"
  "Version number for the Redo package.")

(defvar last-buffer-undo-list nil
  "The head of buffer-undo-list at the last time an undo or redo was done.")
(make-variable-buffer-local 'last-buffer-undo-list)

(make-variable-buffer-local 'pending-undo-list)

;; Emacs 20 variable
(defvar undo-in-progress)

(defun redo (&optional count)
  "Redo the the most recent undo.
```

```
Prefix arg COUNT means redo the COUNT most recent undos.
If you have modified the buffer since the last redo or undo,
then you cannot redo any undos before then."
  (interactive "*p")
  (if (eq buffer-undo-list t)
      (error "No undo information in this buffer"))
  (if (eq last-buffer-undo-list nil)
      (error "No undos to redo"))
  (or (eq last-buffer-undo-list buffer-undo-list)
 ;; skip one undo boundary and all point setting commands up
 ;; until the next undo boundary and try again.
      (let ((p buffer-undo-list))
    (and (null (car-safe p)) (setq p (cdr-safe p)))
    (while (and p (integerp (car-safe p)))
      (setq p (cdr-safe p)))
    (eq last-buffer-undo-list p))
      (error "Buffer modified since last undo/redo, cannot redo"))
  (and (or (eq buffer-undo-list pending-undo-list)
       (eq (cdr buffer-undo-list) pending-undo-list))
       (error "No further undos to redo in this buffer"))
  (or (eq (selected-window) (minibuffer-window))
      (message "Redo..."))
  (let ((modified (buffer-modified-p))
    (undo-in-progress t)
    (recent-save (recent-auto-save-p))
    (old-undo-list buffer-undo-list)
    (p (cdr buffer-undo-list))
    (records-between 0))
 ;; count the number of undo records between the head of the
 ;; undo chain and the pointer to the next change.  Note that
 ;; by `record' we mean clumps of change records, not the
 ;; boundary records.  The number of records will always be a
 ;; multiple of 2, because an undo moves the pending pointer
 ;; forward one record and prepend a record to the head of the
 ;; chain.  Thus the separation always increases by two.  When
 ;; we decrease it we will decrease it by a multiple of 2
 ;; also.
    (while p
      (cond ((eq p pending-undo-list)
         (setq p nil))
        ((null (car p))
         (setq records-between (1+ records-between))
         (setq p (cdr p)))
        (t
         (setq p (cdr p)))))
 ;; we're off by one if pending pointer is nil, because there
 ;; was no boundary record in front of it to count.
    (and (null pending-undo-list)
     (setq records-between (1+ records-between)))
 ;; don't allow the user to redo more undos than exist.
 ;; only half the records between the list head and the pending
 ;; pointer are undos that are a part of this command chain.
    (setq count (min (/ records-between 2) count)
      p (primitive-undo (1+ count) buffer-undo-list))
    (if (eq p old-undo-list)
            nil ;; nothing happened
 ;; set buffer-undo-list to the new undo list.  if has been
 ;; shortened by `count' records.
      (setq buffer-undo-list p)
 ;; primitive-undo returns a list without a leading undo
 ;; boundary.  add one.
      (undo-boundary)
 ;; now move the pending pointer backward in the undo list
 ;; to reflect the redo.  sure would be nice if this list
 ;; were doubly linked, but no... so we have to run down the
 ;; list from the head and stop at the right place.
      (let ((n (- records-between count)))
    (setq p (cdr old-undo-list))
    (while (and p (> n 0))
      (if (null (car p))
          (setq n (1- n)))
      (setq p (cdr p)))
```

```
      (setq pending-undo-list p)))
      (and modified (not (buffer-modified-p))
       (delete-auto-save-file-if-necessary recent-save))
      (or (eq (selected-window) (minibuffer-window))
      (message "Redo!"))
      (setq last-buffer-undo-list buffer-undo-list)))

  (defun undo (&optional arg)
    "Undo some previous changes.
  Repeat this command to undo more changes.
  A numeric argument serves as a repeat count."
    (interactive "*p")
    (let ((modified (buffer-modified-p))
      (recent-save (recent-auto-save-p)))
      (or (eq (selected-window) (minibuffer-window))
      (message "Undo..."))
      (or (eq last-buffer-undo-list buffer-undo-list)
   ;; skip one undo boundary and all point setting commands up
   ;; until the next undo boundary and try again.
      (let ((p buffer-undo-list))
        (and (null (car-safe p)) (setq p (cdr-safe p)))
        (while (and p (integerp (car-safe p)))
          (setq p (cdr-safe p)))
        (eq last-buffer-undo-list p))
      (progn (undo-start)
             (undo-more 1)))
      (undo-more (or arg 1))
   ;; Don't specify a position in the undo record for the undo command.
   ;; Instead, undoing this should move point to where the change is.
   ;;
   ;;;; The old code for this was mad!  It deleted all set-point
   ;;;; references to the position from the whole undo list,
   ;;;; instead of just the cells from the beginning to the next
   ;;;; undo boundary.  This does what I think the other code
   ;;;; meant to do.
      (let ((list buffer-undo-list)
            (prev nil))
        (while (and list (not (null (car list))))
          (if (integerp (car list))
              (if prev
              (setcdr prev (cdr list))
   ;; impossible now, but maybe not in the future
                (setq buffer-undo-list (cdr list))))
          (setq prev list
                list (cdr list))))
      (and modified (not (buffer-modified-p))
       (delete-auto-save-file-if-necessary recent-save)))
    (or (eq (selected-window) (minibuffer-window))
        (message "Undo!"))
    (setq last-buffer-undo-list buffer-undo-list))
```

帮助系统

使用 Emacs 的过程中，您随时可以获取帮助

M-x help-with-tutorial

`C-h t` Emacs快捷指南

M-x info-emacs-manual

`C-h r` Emacs使用手册

M-x info

`C-h i` 在线帮助

其它

表 **25.2. Emacs**帮助系统

C-h a	M-x apropos-command	搜索命令
C-h f	M-x describe-function	函数说明
C-h v	M-x describe-variable	变量说明
C-h k	M-x describe-key	键绑定说明
C-h c	M-x describe-key-briefly	键绑定说明
C-h w	M-x where-is	查找键绑定

- 使用帮助时，可能会分割出其它窗格。 `C-x 1` 关闭其它窗格 详见“窗格和缓冲区管理”一节

基本操作

针对文件及编辑器的一些操作，绝大多数软件中，这类操作都安排在 文件 菜单里面。

表 **25.3. Emacs** 基本操作

C-x C-c	M-x save-buffers-kill-emacs	保存退出
C-x C-z	M-x iconify-or-deiconify-frame	挂起（最小化）
C-x C-f	M-x find-file	打开文件、目录
C-x C-r	M-x find-file-read-only	以只读模式打开
C-x i	M-x insert-file	插入文件
C-x C-s	M-x save-buffer	保存
C-x s	M-x save-some-buffers	询问，保存所有未保存的缓冲区
C-x C-w	M-x write-file	另存为文件
C-x RET r	M-x revert-buffer-with-coding-system	以指定编码读取文件
C-x RET f	M-x set-buffer-file-coding-system	以指定编码保存文件
	M-x revert-buffer	恢复到原始状态

数字参数

Emacs 中可以使用 Ctrl+u 向命令传递参数。例如用数字作为参数，指定命令运行的次数

`C-u (#) command`

M-x universal-argument （通用参数）

例如：

```
C-u 10 C-f              向前10个字符
C-u 10 M-x forward-char
```

`M-(#) (command)`

negative-argument （负参数）

`M-[1-9] 快速参数`

digit-argument （数字参数）

基础编辑

几乎所有编辑器都具有的基础功能。

使用 Readline 控件的程序，例如 bash ；以及其它使用 Emacs风格 键绑定的程序，也使用基本相同的功能键。如果熟悉 bash 的快捷键，这些绑定您一定驾轻就熟

> 注意：这里只是一个列表，更详细的介绍，请参阅《Emacs 快捷指南》 C-h t

定位

表 **25.4. Emacs** 定位

	向前	向后	向下	向上
卷屏	C-v	M-v		
字符	C-f	C-b	C-n	C-p
单词	M-f	M-b		
行	C-a	C-e	移动到行首或行尾，不能跨行	
句	M-a	M-e		
段落	M-{	M-}		
缓冲区	M-<	M->	移动到缓冲区起始或结束	
行号	M-g g	M-g M-g	M-x goto-line	按行号跳转
字符位置	M-x goto-char	按字符跳转		

其它：

`C-M-l` **(M-x reposition-window)**

将当前行卷至页面中部

`C-l` **(M-x recenter)**

刷新页面，将将当前行卷至页面中部 （使用数字参数指定行）

`M-r M-x` **(move-to-window-line)**

移动光标至页面中间的行 （使用数字参数指定行）

删除

表 25.5. Emacs 删除

	向前	向后		
字符	C-d	M-x delete-char	DEL	M-x delete-backward-char
单词	M-d	M-x kill-word	C-Delete / M-DEL	M-x backward-kill-word
行	光标至行末	C-k	M-x kill-line	
整行	C-S-backspace	M-x kill-whole-line		
按表达式删除	C-M-k	M-x kill-sexp		
区块	C-w	M-x kill-region		
空白	删除连续空格	M-x delete-horizontal-space		

> 注意：上表中 DEL 实际按键为 Backspace ，PC 中只有 Delete 键，而没有 DEL 键，Emacs 把 Backspace 映射为 DELbackspace 实际按键也为 Backspace类似的，Emacs 把 PC 的 Enter 键映射为 RET ; 而 RET 实际为 C-m

可能您注意到了，Emacs 进行删除时有两种处理方法， delete 和 kill

kill

比较类似于 剪切 ，剪切掉的内容被依次放入 剪切队列 kill-ring ，可以召回。

delete

就是 删除 了，删除掉的内容并不能召回。但是可以通过 M-x undo 撤消删除。

复制

`M-w` **(M-x kill-ring-save)**

将内容放入 剪切队列 kill-ring

`C-w`

剪切

粘贴

`C-y` **(M-x yank)**

从 剪切队列 kill-ring 中召回最后一次放入的内容

`M-y` **(M-x yank-pop)**

从 剪切队列 kill-ring 中按后进先出的顺序，依次召回

- 这个命令只能在 M-x yank 或者 M-x yank-pop 之后使用。也就是说，只能 C-y 后 M-y ，M-y 可以连续多次。

撤消

`C-/` **(M-x undo)**

撤消之前的修改

`C-_` **(M-x undo)**

`C-x u` **(M-x advertised-undo)**

advertised-unde 是 undo 命令的一个别名

- 为了减少 undo 的次数，每插入20个字符，视为一个 undo 的单位。

重做

安装 redo.el 扩展,并在配置文件中添加如下内容

```
;;---------- redo
;; 读取扩展
(require 'redo)
;; 设置 Redo 的键绑定
(global-set-key ( kbd "C-.") 'redo)
```

区块编辑

如何选中区块，可以参考“区块选择”一节

需要补充的是，完成区块选择时，实际定义了两种区块 ：

```
---XXXXXXxxx
xxxXXXXXXxxx
xxxXXXXXXxxx
xxxXXXXX---
```

---代表没有被选中的区域

连续区块为标记和光标点之间连续的区块；字符 X 和 x 均为连续区块

矩形区块为标记和光标点之间矩形的区块；大写字符 X 为矩形区块

表 **25.6. Emacs** 区块编辑

连续区块		
C-SPC	M-x set-mark-command	在光标点处设置标记
C-@	同上	建议使用 C-t
M-@	M-x mark-word	在单词结尾处设置标记
M-h	M-x mark-paragraph	选中段落
	M-x mark-end-of-sentence	在句末设置标记
C-x h	M-x mark-whole-buffer	整个缓冲区
C-x C-x	M-x exchange-point-and-mark	交换标记和光标点
C-w	M-x kill-region	剪切区块
M-w	M-x kill-ring-save	复制区块
C-y	M-x yank	粘贴区块
M-y	M-x yank-pop	队列粘贴
矩形区块		
C-x r k	M-x kill-rectangle	删除矩形区块
C-x r y	M-x yank-rectangle	粘贴上一次删除的矩形区块
C-x r t	M-x string-rectangle	用指定字符填充
C-x r o	M-x open-rectangle	用空格插入
C-x r c	M-x clear-rectangle	用空格填充
C-x r r	M-x copy-rectangle-to-register	拷贝到寄存器中

搜索和替换

Emacs中，默认使用 增量搜索 ：在搜索对话模式中输入关键词的同时，Emacs 就开始进行搜索，随着关键字的输入，不断的缩小搜索范围

而传统的非增量搜索，则是关键词输入后，再进行搜索。

表 **25.7. Emacs** 搜索

增量搜索		
C-s	M-x isearch-forward	向前增量搜索
C-r	M-x isearch-backward	向后增量搜索
C-M-s	M-x isearch-forward-regexp	正则表达式向前增量搜索
C-M-r	M-x isearch-backward-regexp	正则表达式向后增量搜索
询问替换		
M-%	M-x query-replace	询问替换
C-M-%	M-x query-replace-regexp	正则表达式询问替换
搜索		
	M-x search-forward	向前搜索
	M-x search-backward	向后搜索
	M-x search-forward-regexp	正则表达式向前搜索
	M-x search-backward-regexp	正则表达式向后搜索
替换		
	M-x replace-string	替换
	M-x replace-regexp	正则表达式替换

- 增量搜索时，关键词会被一直保留。可以直接进行下一次搜索
- 下一次增量搜索，如果之前进行了其它操作，则需要连续两次命令（快捷键），才能召回关键词。
- `C-g` 取消搜索，回到搜索前的位置
- `RET` 结束搜索，停在当前位置
- 可以选中区块后，在区块内进行替换

其它

插入控制字符

使用 `C-q` ，可以在缓冲区插入一个控制字符。例如:

`C-q C-m = ^M`

文本换位

表 **25.8. Emacs** 其它

字符	C-t	M-x transpose-chars
单词	M-t	M-x transpose-words
行	C-x C-t	M-x transpose-lines

将 *TAB* 字符转换为空格

选中需要转换的区域， **M-x untabify**

对齐文本块

选中需要对齐的区域， **M-x indent-region**

窗格和缓冲区管理

多窗格

表 **25.9. Emacs** 窗格

C-x 2	M-x split-window-vertically	分隔出两个垂直窗格，水平分隔线
C-x 3	M-x split-window-horizontally	分隔出两个水平窗格，垂直分隔线
C-x 1	M-x delete-other-window	只保留当前窗格
ESC ESC ESC	M-x keyboard-escape-quit	只保留当前窗格
C-x 0	M-x delete-window	关闭当前窗格
C-x o	M-x other-window	在下一个窗格中激活光标
C-M-v	M-x scroll-other-window	向下卷动下一个窗格，使用负参数可以向上卷动

- 下一个窗格: 垂直分隔，则先左后右；水平分隔，则先上后下。如果窗格还有子窗格，则先遍历其子窗格后，再遍历其它窗格，以此递归。

多缓冲区

Emacs 中，打开新的缓冲区，原有缓冲区并不会关闭

表 **25.10. Emacs** 缓冲区

C-x C-b	M-x list-buffers	查看缓冲区列表
C-x b	M-x switch-to-buffer	切换到其它缓冲区
C-x k	M-x kill-buffer	关闭当前缓冲区

- 切换到其它缓冲区时，默认上一次使用的缓冲区(可以用 TAB 补全)
- 使用多窗格时，缓冲区操作只对当前窗格有效
- 建议使用 ibuffer.el 这个扩展。 Emacs 自带，在配置文件中添加如下语句

```
;;========== ibuffer
(require 'ibuffer)
(global-set-key ( kbd "C-x C-b ")' ibuffer)
```

- 另一个缓冲区列表的扩展（Emacs 自带）

```
;;CRM bufer list
(global-set-key "\C-x\C-b" 'electric-buffer-list)
```

寄存器管理

寄存器用于存贮内容，在需要时取出，插入缓冲区。

Emacs 的寄存器使用单个字符命名，可以存贮两种内容 ：

光标位置和窗口状态

表 **25.11. Emacs** 寄存器

C-x r SPC (寄存器名)	M-x point-to-register	存贮光标位置
C-x r w (寄存器名)	M-x window-configuration-to-register	保存当前窗口状态
C-x r f (寄存器名)	M-x frame-configuration-to-register	保存所有窗口状态
C-x r j (寄存器名)	M-x jump-to-register	光标跳转
C-x j (寄存器名)	略……	同上

文本和数字

表 **25.12. Emacs** 寄存器**2**

C-x r s (寄存器名)	M-x copy-to-register	将连续区块拷贝到寄存器中
C-x r r (寄存器名)	M-x copy-rectangle-to-register	将矩形区块拷贝到寄存器中
C-u (数字) C-x r n (寄存器名)	M-x number-to-register	将数字拷贝到寄存器中
C-x r i (寄存器名)	M-x insert-register	在缓冲区中插入寄存器内容

- **M-x view-register** 查看寄存器内容
- **M-x list-registers** 查看寄存器列表
- 寄存器中的矩形区块，以矩形区块的方式插入到缓冲区中。 见“区块编辑”一节
- 也可以将文件插入到寄存器中 (set-register ?寄存器名称 '(file . 文件名)) ，示例

```
M-x lisp-interaction-mode 进入交互模式，输入如下 Lisp 代码：
(set-register ?e '(file . "~/.emacs"))(光标)移动此外， C-j 求值。
M-x list-registers 查看寄存器列表，多了寄存器 e ： Register e contains the file
```

书签管理

Emacs 可以在当前位置创建一个书签，以便能够快速的返回。

与存储光标位置的寄存器略有不同

- 书签可以使用单词来命名，而不限于一个字符。起一个容易记住的名字
- 退出 Emacs 后，书签不会消失，下次还可以使用

表 25.13. **Emacs** 书签

C-x r m (name)	M-x bookmark-set	设置书签
C-x r b (name)	M-x bookmark-jump	跳转到书签
C-x r l	M-x bookmark-bmenu-list	书签列表
	M-x bookmark-delete	删除书签
	M-x bookmark-load	读取存储书签文件

- 书签默认存储在 `~/.emacs.bmk` 文件中
- 在配置文件中，可以设置书签存储的文件

```
;; 书签文件的路径及文件名
(setq bookmark-default-file "~/.emacs.d/.emacs.bmk")

;; 同步更新书签文件 ;; 或者退出时保存
(setq bookmark-save-flag 1)
```

Shell 模式

`M-x shell` 进入 Shell 模式，可以完成一些简单的工作。不过有些情况下，输出会有一些问题

事实上，这是 Emacs 自带的终端。它与 bash 和 sh 的兼容比较好，而 fish 之类比较现代的 Shell，在 Emacs 终端里的效果则很差

需要注意的是， readline-bash 的绑定 C-p C-n ，在 Emacs 终端需要使用 M-p M-n 。其它的键绑定，也以 Emacs 为准

宏

记录一系列操作，在需要的时候运行

例如给一个单词加 " ，可以分解为以下操作:

```
M-b 移动到词首
"
M-f 移动到词尾
"
```

这种重复的操作往往需要经常执行，手动未免太没有效率。我们可以把这些操作制作成宏，然后运行这个宏

当然，这只是最简单的宏。结合正则表达式进行匹配，以宏进行操作，可以完成许多复杂的操作

定义与运行

表 25.14. Emacs 宏

开始录制	C-x ((M-x kmacro-start-macro)	F3 (M-x kmacro-start-macro-or-insert-counter)
结束录制	C-x) (M-x kmacro-end-macro)	F4 (M-x kmacro-end-or-call-macro)
播放	C-x e (M-x kmacro-end-and-call-macro)	

宏队列

与 剪切队列 类似，Emacs 中也有 宏队列 的概念： 当一个新的宏被定义，原有的宏并不消失，只是在宏队列中的位置被挤到后面。

`C-x C-k` 进入宏队列，以下的操作可以在宏队列中连续进行

表 **25.15. Emacs** 宏队列

基本操作		
C-n	M-x kmacro-cycle-ring-next	下翻
C-p	M-x kmacro-cycle-ring-previous	上翻
C-d	M-x kmacro-delete-ring-head	删除当前宏
C-k	M-x kmacro-end-or-call-macro-repeat	运行当前宏
命名与保存		
n (name)	M-x kmacro-name-last-macro	命名
b	M-x kmacro-bind-to-key	绑定
	M-x insert-kbd-macro	在缓冲区中插入宏定义
宏编辑器		
C-e	M-x kmacro-edit-macro	编辑
e	M-x edit-kbd-macro	编辑指定名称的宏
l	M-x kmacro-edit-lossage	
询问执行		
q	M-x kbd-macro-query	在播放宏时，将进行询问确认
计数器		
C-i	M-x kmacro-insert-counter	将宏计数器的数值插入缓冲区
C-c	M-x kmacro-set-counter	为宏计数器设置一个数值
C-a	M-x kmacro-add-counter	给宏计数器添加一个前缀参数
C-f	M-x kmacro-set-format	给宏计数器指定一个将要插入的特殊值

- 保存为文件，使用 **M-x load-file** 加载
- 保存到配置文件中，启动时加载

文件管理

文件管理

`C-x d` **(M-x dired)**

进入 Dired 列表模式

`C-x C-d` **(M-x list-directory)**

获取文件列表（简洁）

`C-x C-f` **(M-x find-file)**

打开文件，没有文件名则打开目录

定位、查看

表 **25.16. Emacs** 文件管理

	向下	向上	
文件	n	p	
C-n	C-p		
SPC	DEL	上一级	
目录	>	<	^
已标记	M-}	M-{	

g	刷新
s	切换名称/日期排序方式
i	在当前窗口插入一个子目录
v	查看当前文件
y	查看当前文件类型
=	比较

标记

普通标记	m	标记（显示为字符 *)
t	反向标记	
u	取消标记	
U	取消所有标记	
* /	标记文件夹	
	标记所有可执行文件	
* @	标记所有符号连接	
* c	改变标记的符号	
% m	根据正则表达式标记文件	
% g	根据正则表达式标记文件内容	
删除标记	d	标记为删除(显示为字符 D)
~	将备份文件标记为删除	
#	将存盘文件标记为删除	
& d	根据正则表达式标记删除	
x	执行删除	

操作

RET	在新缓冲区打开文件	
o	在另一个窗格打开	
C-o	在另一个窗格后台打开	
C-x C-f	新建文件	
+	新建目录	
C-x C-q	将文件列表设为只读	
可以结合 * 标记批量进行	D	删除文件
C	拷贝	
R	重命名/移动	
O	改变用户	
G	改变群组	
M	改变权限	
S	符号链接	
H	硬链接	
Z	压缩	
T	touch	
w	复制文件名	
k	删除行(刷新后恢复)	

服务器模式

由于各种原因，Emacs 启动比较耗时。可以启动一个 Emacs 的守护进程

```
emacs --daemon
```

然后通过 emacsclient 来连接服务器

```
emacsclient  -t  --alternate-editor jed  file
```

- *-t* 在当前控制台打开 emacs 窗口
- *--alternate-editor jed* 如果不能连接到 emacs 服务器，则使用 jed 编辑器

也可以使用 Emacs 服务器模式，`M-x server-start` 或者在配置文件中添加 *(server-start)* 启用 Emacs 服务器，使用 emacs-client 连接。

大纲模式

Emacs 的大纲模式(Outline mode)，是一个十分有用的模式。如果工程规模比较大，你应该用大纲来组织它。

大纲模式通常作为辅模式使用，**M-x outline-minor-mode**启用。

大纲模式可以根据代码的语法对结构进行识别，但是这种自动模式工作的不是很好，而且不够灵活

另一种工作方式是查找特定的字串，来组织文档的结构。这种工作方式是可控制的，不过需要手动加入这些作为结构标识的字串。似乎有点麻烦，但是对于严谨的构思来说，你是需要列一个提纲的

大纲模式默认以行首的 `*` 作为结构标识，每多一个 `*` 就是下一级分支。

大纲模式默认的键绑定太过复杂，在配置文件中添加以下语句，将所有大纲模式操作的前缀键改为 Ctrl+o

```
(setq outline-minor-mode-prefix [(control o)])
```

试着在文档中随便加入几个这样的标识

```
*主干一
**分支一
**分支二
*主干二
```

进行一些简单的操作

C-o C-a	全部显示
C-o C-t	显示主干
C-o C-q	全部隐藏
C-o C-i	显示下一级分支
C-o C-k	显示分支
C-o C-e	显示节点
C-o C-d	隐藏分支

定制结构标识

为了不影响代码的功能，通常要把结构标识放到注释中，这样作有可能会带来不便,例如在 XML 环境中就要像这样使用：

```
<!--
*结构标识
-->
```

这样会给你的工作制造出一些混乱，并且大纲模式是以行为单位进行识别的，也就是说`<!--`属于上一个分支

好在大纲模式可以通过正则表达式来定义结构标识，你可以把结构标识定义为下面这种形式

```
<!--*结构标识-->
```

通过设置`outline-regexp`定制标识：

```
setq outline-regexp "<!\\-\\-\\*+
```

> ❶ Elisp 语法中，`\` 有特殊意义，所以脱字符要用`\\`表示
>
> ❷ 正则表达式中 `-` 有特殊意义，所以前面要加脱字符
>
> ❸ `*+` 表示一个或多个`*`

配置

完整的配置：

例 **25.2. emacs** 大纲模式

```
(setq outline-minor-mode-prefix [(control o)])

(require 'docbook-xml-mode)
(add-hook 'docbook-xml-mode-hook
      (function (lambda ()
                  (setq outline-regexp "<!\\-\\-\\*+")
              (outline-minor-mode)
              (hide-body)
                  )))
```

> ❶ 更改大纲模式前缀键
>
> ❶ 加载 docbook-xml-mode
>
> ❸ 添加 docbook-xml-mode 钩子，运行下面代码[47]
>
> ❹ 将`<!--*`识别为标识
>
> ❺ 启动大纲模式作为辅模式
>
> ❻ 隐藏所有内容，只显示主干

操作列表

显示、隐藏

	全部显示	显示分支	隐藏			
全局	show-all	C-o C-a	hide-body	C-o C-t	hide-sublevels	C-o C-q❶
分支	show-subtree	C-o C-s	hide-leaves	C-o C-l	hide-subtree	C-o C-d
show-branches	C-o C-k					
show-children	C-o C-i					
节点	show-entry	C-o C-e	hide-entry	C-o C-c		
其它分支	hide-other	C-o C-o				

> ❶ 可以带数字参数，如 `M-2 C-o C-q` 显示第2层子结构

移动

	向前	向后		
全局	outline-next-visible-heading	C-o C-n	outline-previous-visible-heading	C-o C-p
同级	outline-forward-same-level	C-o C-f	outline-backward-same-level	C-o C-b
返回上一级	outline-up-heading	C-o C-u		

> [47] `docbook-xml-mode.el` 中定义 docbook-xml-mode-hook，模式启动时运行钩子代码

在 Emacs 中使用 sdcv

在 Emacs 配置文件中加入以下代码

```
(global-set-key (kbd "C-c d") 'kid-sdcv-to-buffer)
(defun kid-sdcv-to-buffer ()
  (interactive)
  (let ((word (if mark-active
                  (buffer-substring-no-properties (region-beginning) (region-end))
                  (current-word nil t))))
    (setq word (read-string (format "Search the dictionary for (default %s): " word)
                            nil nil word))
    (set-buffer (get-buffer-create "*sdcv*"))
    (buffer-disable-undo)
    (erase-buffer)
    (let ((process (start-process-shell-command "sdcv" "*sdcv*" "sdcv" "-n" word)))
      (set-process-sentinel
       process
       (lambda (process signal)
         (when (memq (process-status process) '(exit signal))
           (unless (string= (buffer-name) "*sdcv*")
             (setq kid-sdcv-window-configuration (current-window-configuration))
             (switch-to-buffer-other-window "*sdcv*")
          (local-set-key (kbd "d") 'kid-sdcv-to-buffer)
             (local-set-key (kbd "q") (lambda ()
                                        (interactive)
                                        (bury-buffer)
                                        (unless (null (cdr (window-list))) ; only one win
                                          (delete-window)))))
           (goto-char (point-min))))))))
```

> ❶ 如果选中区域则查询区域内容，否则查询当前光标所在单词。查询结果显示在一个叫做 *sdcv* 的缓冲区
>
> ❷ 在 *sdcv* 里面按 q，将它隐藏到缓冲区列表的结尾
>
> ❸ 在 *sdcv* 里面按 d 查询当前单词

Windows 下字体设置

```
(set-default-font "Verdana-10")
(if window-system
   (set-fontset-font (frame-parameter nil 'font)
      'unicode '("simsun" . "unicode-bmp"))
)
```

> ❶ 英文字体
>
> ❷ 中文字体

版本

在 Linxu 系统中，Emacs 的最新版本通常为 emacs-snapshot、emacs-cvs

Emacs for Windows 请到这里下载，推荐“patched”版本

第 26 章 正则表达式

目录

简介

对于文本内容的处理，通常使用交互方式，手工调整；但如果你对源文本比较了解，则可以采用自动化的批量处理方式，这种方式效率高、迅速快

批量处理，要求根据一定规则，匹配源文本中的字符，转换为目标文本，这就要用到正则表达式

最简单的例子，使用 `regular` 进行匹配，结果如下：

```
`regular` expression
```

正则表达式有许多变种：glob 表达式、基本正则表达式、perl 正则表达式、emacs 正则表达式……

如“通配符”一节中介绍的为最简单的 glob 表达式

运算优先级

正则表达式与数学表达式的不同在于，数学表达式执行数学运算，而正则表达式执行字符运算；相同的是，它们都按一定的优先级进行运算

运算符	操作
\	转义符
()	捕获、匹配、断言
[]	字符类
*+?	限定符
{}	范围
^$	位置和顺序
|	或

转义符

如果源文本中出现了正则表达式中的运算符，如 `(` ，使用 `(` 无法匹配下列文本中的括弧，这时要使用 `\` 进行转义。用 `\(` 匹配[48]:

```
`(`regular expression)
```

在文本中匹配“运算优先级”一节中的所有运算符，都要用这种形式：

```
\运算符
```

在文本中匹配 `\` 本身，要用 `\\`

非运算符前使用 `\` ，则有特殊的意义，例如 `\n` 匹配一个换行符。常用转义字符：

转义字符	涵义	
常规匹配	.	匹配除换行符以外的任意字符
\w	匹配字母或数字或下划线或汉字	
\s	匹配任意的空白符	
\d	匹配数字	
\b	匹配单词的开始或结束，在字符类里代表退格	

^	匹配字符串的开始，在字符类里表示”非“	
$	匹配字符串的结束	
反向匹配	\W	匹配任意不是字母，数字，下划线，汉字的字符
\S	匹配任意不是空白符的字符	
\D	匹配任意非数字的字符	
\B	匹配不是单词开头或结束的位置	
[^aeiou]	匹配除了 aeiou 这几个字母以外的任意字符	
特殊字符	\a	报警字符(打印它的效果是电脑嘀一声)
\t	制表符，Tab	
\r	回车	
\v	垂直制表符	
\f	换页符	
\n	换行符	
\e	Escape	
\0nn	ASCII 代码中八进制代码为 nn 的字符	
\xnn	ASCII 代码中十六进制代码为 nn 的字符	
\unnnn	Unicode 代码中十六进制代码为 nnnn 的字符	
\cN	ASCII 控制字符。比如 \cC 代表 Ctrl+C	
\A	字符串开头(类似^，但不受处理多行选项的影响)	
\Z	字符串结尾或行尾(不受处理多行选项的影响)	
\z	字符串结尾(类似$，但不受处理多行选项的影响)	
\G	当前搜索的开头	

> [48] 在 Emacs 和 Vim 正则表达式中正好反过来，使用 `\(` 表示分组，用 `(` 匹配字符

字符类

要想匹配数字、字母、空白很容易，因为已经有了对应这些字符集合的转义符，但是如果你想匹配没有预定义的字符集合(比如元音字母 a、e、i、o、u)，应该怎么办？

正则表达式中允许你自定义字符类，在方括号里列出它们就可以了

```
[aeiou]
```

预定义的字符集合，也可以用字符类表示，如 `\d` 等价于 `[0-9]`

有些运算符，在字符类中使用会有另一种意义，例如 `^` 表示“字符串开始”，但在字符类中却表示 “非”，以 `expression` 为例，使用 `[exp]` 匹配：

```
`exp`r`e`ssion
```

使用 `[^exp]` 匹配(字符串中非 e、x、p 的字符)：

```
exp`r`e`ssion`
```

而使用 `^[exp]` 匹配(以 e、x 或 p 起始的字符串)：

```
`e`xpression
```

限定符

在上一小节中的表格中，我们知道 `.` 可以匹配除换行符以外的任意字符，使用 `.` 匹配下列文本：

```
expression
```

但是 `.` 每次只匹配一个字符，如果想一次匹配多个，则要使用限定符

限定符	作用
*	匹配零次或多次
+	匹配一次或多次
?	匹配零次或一次
{3}	匹配三次
{3,5}	匹配三到五次
{3,}	匹配三次或以上

下面通过实例了解限定符的区别。 `es` 的匹配结果

```
expr`es`sion
```

`es+` 的匹配结果(e，一个或多个 s)

```
expr`ess`ion
```

`es*` 的匹配结果(e，零或多个 s)

```
`e`xpr`ess`ion
```

`es?` 的匹配结果(e，零或一个 s)

```
`e`xpr`es`sion
```

贪婪与懒惰

使用限定符进行匹配时，默认匹配尽可能多的字符。无论用 `.*` 还是 `.+` 匹配下列文本，都会匹配全部

```
`expression`
```

这种方式称为“贪婪模式”。在限定符之后加 `?` 则匹配尽可能少的字符，称为“懒惰模式”[49]

例如，使用贪婪模式 `a.+b` 匹配：

```
`aaabab`
```

使用懒惰模式 `a.+?b` 匹配：

```
`aaab`ab
```

> [49] `.+` 匹配一个或多个任意字符，在贪婪模式中，它匹配尽可能多的字符；而懒惰模式中(`.+?`)，则只匹配一个字符； `.{3,5}` 在贪婪模式中尽可能匹配5个字符，在懒惰模式中(`.{3,5}?`)只匹配3个字符； `?` 和 `*` 这样可以匹配零次的限定符，在懒惰模式下不匹配任何字符(`.*?` 、 `.??`)

分支条件

`|` 表示“或”，使用它进行分支选择

例如 `[a-z]+|\d+` 匹配单词或数字：

```
expression 123
```

分组、捕获

分组

使用 `(表达式)` 对表达式进行分组，例如使用 `(\d{3}\.){2}` 匹配下面例子中的数字：

```
abc`123.456.`def
```

`\d{3}` 表示三个数字， `(\d{3}\.)` 表示三个数字加“ `.` ”为一组， `{2}` 表示这一组内容重复两次

捕获

在对表达式进行分组的时候，会捕获文本到自动命名的组里，使用 `\1 \2 ……` 后向引用组

例如用 `([a-z]*)\ (\d*)` 匹配下列文本， `([a-z]*)` 为 `\1` 组， `(\d*)` 为 `\2` 组

```
kardinal 1234567
```

使用 `\2\ \1` 替换 `([a-z]*)\ (\d*)` ，可以改变两个字符串的顺序

```
1234567 kardinal
```

如果分组较多，计数可能会不太方便，可以给分组指定名称，例如：

```
(?<**name**>[a-z]*)\ (?<**num**>\d*)
      \k<**num**>\ \k<**name**> (?#使用“`\k<name>`”后向引用)
```

使用 `(?:表达式)` ，则只是分组，而不捕获，下面例子中， `(\d*)` 为 `\1` 组

```
(?:[a-z]*)\ (\d*)
```

零宽断言

目前为止，我们学到的正则表达式匹配，都是有“宽度”的，使用 `\w+。` 匹配下面文本，会将 `。` 一同匹配：

```
regular。
expression。
```

如果不想匹配符号，只匹配一个位置，就要用到“零宽断言”(匹配宽度为零，满足一定的 条件/断言)，零宽断言使用 **(?=**表达式**)** 的语法，例如 `\w+(?=。)` ，其中 `(?=。)` 表示 `。` 前面的位置(先行断言)

```
`regular`。
`expression`。
```

如果需要匹配后面的位置，如：

```
。`regular`
。`expression`
```

则要用到后发断言 `(?<=。)` ，使用 `(?<=。)\w+` 得到上面的匹配结果

使用 `(?<=<b>).*(?=</b>)` 匹配标签中的内容

```
<b>`粗体`</b>
```

负向零宽断言

负向零宽断言 **(?!**表达式**)** 也是匹配一个零宽度的位置，不过这个位置的“断言”取表达式的反值，例如 `(?!表达式)` 表示 `表达式` 前面的位置，如果 `表达式` 不成立 ，匹配这个位置；如果 `表达式` 成立，则不匹配：

```
`expression`
`expression`，
`expression`；
expression。
```

以上为使用 `.+n(?!。)` 的匹配结果。注意与 `.+n[^。]` 匹配的区别

```
expression
`expression，`
`expression；`
expression。
```

同样，负向零宽断言也有“先行”和“后发”两种，负向零宽后发断言为 **(?<!表达式)**

使用 `(?<![</])para(?!>)` 匹配下面文本

```
<para>`para`表示一个段落</para>
```

- `(?<![</])` 表示 `para` 左边不能为 `<` 或 `/` ；`(?!>)` 表示 `para` 右边不能为 `>`

第 27 章 docbook 指南

目录

XML 简介

在学习 DocBook 之前，我们需要先了解一下 XML，因为 DocBook 是 XML 的一个 DTD（文档类型定义）

XML 是一种被设计用来存储、交换数据的通用标记语言

为了使它更加的通用，XML 的元标记不具有意义，XML 使用 DTD 赋予某一组标记特定的意义

为了便于自动处理，它只包含内容而不包含样式定义，XSL 便是这样一种自动处理的机制，它将根据特定规则将 XML 转换为可以定义样式的格式

Html 语言不具备以上特点，我们用它对比说明，一份完整的 Html 文档就是由许多这样的标记嵌套而成：

```
<html>
    <head>头部</head>
    <body>
        <p>这是一个段落，这里是 <b>粗体</b>    </p>
    </body>
</html>
```

- Html 语言的标记，都有具体的意义，像 `<p>` 表示这是一个段落
- Html 语言的标记，还可以直接定义内容的样式。比如加粗某处文字，使用标记 `<b>粗体</b>` 。 `<b>` 是开始标记，它告诉浏览器，从这个标记开始，后面内容用粗体显示； `</b>` 是结束标记，它告诉浏览器，粗体显示到这里结束

由于 Html 语言的标记都有具体的意义，都和网页显示有关，所以它也只能用来显示网页。如果在 DTD 中定义 `<b> <p>` 这些标记的意义，XML 也可以显示网页（xhtml）

假设有一段文字，里面提到一个文件名和一个软件名，由于 Html 的标记指定的是样式而不是内容，作为变通，我们可以使用粗体来表示它们，但是不能准确的区分它们。而 XML 定义的是内容，把它们分别定义为 filename 和 application，然后通过 XSL 给它们指定不同的样式，便可以很容易的区分

XML 语法

标记

XML 使用 `< >` 来定义标记

所有的 XML 标记必须结束。 `<para>` 必须有一个与之配对的 `</para>` ；或者使用空标记，例如 `<para/>`

XML 标记大小写敏感。 `<para>` 和 `<PARA>` 是两个不同的标记；在 Docbook 中，所有的标记都是小写的

标记必须正确的嵌套，不允许相互嵌套

```
<section><para></section></para>  错误

<section><para></para></section>  正确
```

元素

XML 使用开始标记和结束标记来定义元素。例如 `<para>段落</para>` ；或者使用空标记，例如 `<xref linkend="ln"/>`

元素可以带有一个或多个属性，也可以不带属性。属性值必须加引号，例如 `<para id="p1">`

处理指令

使用 `<? ?>` 括起来，例如： `<?linebreak?>` 插入一个断行符参见

文件头部

```
<?xml version='1.0' encoding="UTF-8"?>
<!DOCTYPE article
PUBLIC "-//OASIS//DTD DocBook XML V4.5/zh_cn" "http://www.oasis-open.org/docbook/xml/4.5/
```

> ❶ 处理指令
>
> ❷ XML 的版本是1.0
>
> ❸ 文档的编码是 UTF-8
>
> ❹ 开始声明 DTD(文档类型定义)
>
> ❺ 声明文档的根元素是 article
>
> ❻ 公共标识
>
> ❼ 系统标识

实体

给内容指定一个名称，通过名称引用。引用时以 `&` 起始， `;` 结束。

这样定义一个实体

```
<!ENTITY x "XML">
```

输入 `&x;` ，XML 会用 `XML` 替换它

以上是内部实体，可以通过定义外部实体来引用外部文档

```
<!ENTITY docbook SYSTEM "docbook.xml">
```

输入 `&docbook;` ，会将 `docbook.xml` 文件中的内容插入到当前位置

内部实体和外部实体都是普通实体，只能够在 DTD 中定义

```
<?xml version='1.0' encoding="UTF-8"?>
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.5/zh_cn"
               "http://www.oasis-open.org/docbook/xml/4.5/docbookx.dtd"
[
<!ENTITY x "XML">
<!ENTITY docbook SYSTEM "docbook.xml">
]>
```

> ❶ 这里声明 DTD 的方法是引用外部 DTD
>
> ❷ 自己定义的实体为内部 DTD

- 内部 DTD 的定义优先于外部 DTD，如果你定义的实体外部 DTD 中已包含，引用时使用的是你自己的定义

有一些特殊字符不能直接插入 XML 中，例如 `<` 、 `&` 、 `空格字符` [50]，使用 `<` 、 `&` 、 ` ` [51]输入这些字符

> [50] 不管有多个空格，XML 都认为只有一个
>
> [51] 它们预先定义在外部 DTD 中 参见 `/usr/share/xml/docbook/xml-dtd-4.5/ent/isonum.ent`

DocBook 介绍

前面已经讲了，DocBook 是 XML 的一个 DTD[52]。

DocBook 主要用来写文档，尤其适合写科技文档。

- 尽管它的语法不如 reStructuredText 之类文档语言简明清晰，但是它的表现能力足够强大
- 它的结构控制能力十分强悍，对于大型的文档工程，DocBook 几乎是不二的选择
- DocBook 足够通用，它已经成为一种事实上的标准，可以生成多种格式的文档，并为多种文档工具所支持，它特别适合作为原始文档来生成其它格式文档

> [52] 它相当庞大，有400个左右的标记；当然常用的也就几十个

搭建 DocBook 环境

在 Linux 系统中，DocBook 环境主要由以下几个软件包提供

libxml2	XML 解析器
docbook-xml	DocBook 的 DTD 定义
libxslt	XSL 转换程序❶
docbook-xsl	用来处理 DocBook-XML 的样式表

> ❶ xsltproc 这个转换程序就在这个软件包中
>
> 还有两个 JAVA 的转换程序 saxon、 xalan，不推荐

例如，在 Archlinux 中可以这样安装：

```
sudo pacman -S docbook-xml docbook-xsl libxslt libxml2
```

创建 DocBook 文档

这是一个最基本的 Docbook 文档

```
<?xml version='1.0' encoding="UTF-8"?>
<!DOCTYPE article
PUBLIC "-//OASIS//DTD DocBook XML V4.5/zh_cn" "http://www.oasis-open.org/docbook/xml/4.5/
<article>
  <info>
    <title>文章</title>
    <author>作者</author>
    <address>地址</address>
    <copyright><year>2008</year><holder>所有者</holder></copyright>
  </info>
  <sect1>
    <title>标题</title>
    <para>
      内容
    </para>
  </sect1>
</article>
```

> ❶ 文件头，其实不需要深入的了解它，因为它的变动通常很小 留意一下系统支持的 DocBook DTD 最高版本**ls /usr/share/xml/docbook**
>
> ❷ 文档的根元素，要紧跟文件头
>
> ❸ 文档信息
>
> ❹ 文档主体，“分子元素”一节中详细介绍

将上面代码保存为 `docbook.xml`，试着发布

结构元素

DocBook 中的一些元素，在发布的时候会作为条目被收进目录。它们表示的是文档内部的结构，所以它们是结构元素,如下：

```
<set>
  <book>
    <part>
      <chapter>
        <sect1>
          <sect2>
        ...........
          </sect2>
        </sect1>
      </chapter>
    </part>
  </book>
</set>
```

如果 DocBook 的根元素是**article**，那么可以允许这种结构：

```
<article>
  <sect1>
    <sect2>
        ...........
    </sect2>
  </sect1>
</article>
```

- 结构元素可以拥有标题，**title**、**titleabbrev**、**subtitle**任选一个，还可以拥有**info**

可以看出，无论是 Book 还是 Article，自 **sect1**[53]而下的结构都是相同的。

如果把**sect1**元素的内容放在一个单独的文件中，那么无论是 Book 还是 Article 都可以不加改动的引用这个文件，参见“文档工程”一节

以下是常用的结构元素

元素	说明
set	集
book	书
part	部
chapter	章
article	文章
sections	节

> [53] section 与 sect1、sect2……sect5 的区别在于，section 可以无限嵌套，sect1 到 sect5 只能逐级嵌套
>
> 推荐使用 sect1~5，这样可以使文档的结构更加清晰，而且通常5级也够用了；如果超过5级，那样的文章估计也没法读

文档信息

info跟在根元素之后，用来放置一些文件信息

```
<info>
  <title>文章</title>
  <author>作者</author>
  <address>地址</address>
  <copyright><year>2008</year><holder>所有者</holder></copyright>
</info>
```

分子元素

DocBook 最基本的元素是 **para** 和块元素，比它们大的是结构元素，比它们小的是行内元素。

元素	说明	
sect1	节	❶
section	节	❷
para	段落	❸
formalpara	带标题段落	❹

> ❶ sect1-5逐级嵌套
>
> ❷ 可无限嵌套
>
> ❸ 简单段落
>
> ❹ 复杂段落，可以带标题

section其实属于结构元素，它的标题会被收录到目录中。把它们放在这里，因为它们是放进单独文件的最佳元素

块元素

块元素是和**para**同级的元素，它们比较复杂，后面会分别介绍

类别	元素	说明
列表	calloutlist	
bibliolist	书目列表	
glosslist	词汇列表	
itemizedlist	无序列表	
orderedlist	有序列表	
segmentedlist	成分列表	
simplelist	简单列表	
variablelist	定义列表	
告示	caution	小心
important	重要	
note	注意	
tip	提示	
warning	警告	
文本	address	地址
literallayout	纯文本	
programlisting	代码	
screen	文本抓屏	
synopsis	命令、函数纲要	
例子，图片和表格	example	
informalexample		
figure		
informalfigure		
table		
informaltable		
媒体文件	audioobject	音频对象
imageobject	图片对象	
imageobjectco	带 callout 的图片对象	
videoobject	视频对象	
textobject	文本对象	
其他	blockquote	引用
epigraph	题记	

msgset	有关的错误信息
sidebar	侧边栏

行内元素

假如你想用特殊的格式表示诸如短语、用户输入、命令、软件、文件，尽量不要使用emphasis(强调)，它们有专门的元素来表示。

尽管使用了这些元素，得到的效果可能和强调一样是粗体，而且还要考虑使用哪种类型，比较麻烦。

不过等到你想用红色表示命令、绿色表示文件……，这个时候，如果你一直用的是emphasis，你将会为你的草率付出代价。

元素名	含义	示例
abbrev	略写，一般后面带有一个点('.')	<abbr class="abbrev">abbrev</abbr>
emphasis	强调	强调
phrase	短语	短语
quote	用引号引起来	“用引号引起来”
trademark	注册商标	注册商标™
literal	文字的字面含义	文字的字面含义
prompt	提示符	提示符
userinput	用户输入	用户输入
subscript	下标	_{下标}
superscript	上标	上标
application	软件名	软件名
command	命令	命令
envar	环境变量	环境变量
filename	文件名	文件名
database	数据库	数据库
email	电子邮件	<[电子邮件](mailto:%E7%94%B5%E5%AD%90%E9%82%AE%E4%BB%B6)>

特殊字符

一些字符在 XML 中有特殊的含义，只能够通过其实体名称输入

字符	写法	缩写涵义
<	<	less than
>	>	greater than
&	&	ampersand
"	"	quote
'	'	apostrophe
空格		none-break space

- 通常需要使用实体输入的字符包括`<`、`&`、`空格` XML 会将任意数量的空格解析为一个，如果想通过多个空格控制缩进，就要使用` ` `]]>`这个字符序列也不能直接输入，这样输入它`]]>`

也可以不使用实体输入特殊字符，通过**CDATA**直接引用：

```
<![CDATA[ `<&` ]]>
```

列表

最普通的列表为无序列表和有序列表：

```
<itemizedlist>
  <listitem>
    列表项内容，可以使用**para**、**formalpara**等
  </listitem>
</itemizedlist>
```

> ❶ 无序列表；有序列表为**orderedlist**，每个项自动编号
>
> ❷ 列表项

词汇列表

词汇列表每一项包括一个词汇标题和词汇定义，标题粗体，定义缩进。这本身就是一个词汇列表

```
<glosslist>
    <glossentry>
        <glossterm>词汇标题</glossterm>
        <glossdef>
             <para>词汇定义</para>
        </glossdef>
    </glossentry>
</glosslist>
```

> ❶ 词汇列表
>
> ❷ 列表项
>
> ❸ 词汇标题
>
> ❹ 词汇定义
>
> ❺ 词汇定义必须为**para**元素

Callout 列表

在引用的代码块中加记号，并在外部注释。如果你要解释一个比较复杂的概念，**calloutlist**通常是最好的选择

```
<screen>
这里会有一个数字`1`<co id="1"/>的记号
它就是 callout <co id="2"/>
</screen>

<calloutlist>
  <callout arearefs="1">这里是记号`1`的说明</callout>
  <callout arearefs="2">第二个记号的说明</callout>
</calloutlist>
```

> ❶ 使用 `<co id="1"/>` 作记号
>
> ❷ 第二个记号
>
> ❸ calloutlist 列表，每一个项都用数字编号
>
> ❹ callout 元素作为解释，使用 `arearefs=""` 引用

表格

表格更适合用可视化方式来生成；因为它有两个维度，使用线性的标记语言来生成，自然比较麻烦。

DocBook 生成表格虽然麻烦，但是相对 reStructuredText 等使用空间方位生成的方式，却更容易控制

表格分为两种**table**和**informaltable**，区别在于**table**可以有标题，能够被收录到目录中。下面是表格的实例：

表 **27.1.** 表格实例

	表头	
	表底	
单元格一	单元格二	单元格三
11	12	13
	22	23
31		

```
<table>
  <title>表格实例</title>
  <tgroup cols="3">
    <thead><row><entry></entry><entry>表头</entry></row></thead>
    <tfoot><row><entry></entry><entry></entry><entry>表底</entry></row></tfoot>
  <tbody>
    <row>
        <entry>单元格一</entry>
        <entry>单元格二</entry>
        <entry>单元格三</entry>
    </row>
    <row>
    <entry>11</entry>
    <entry>12</entry>
    <entry>13</entry>
    </row>
    <row>
    <entry></entry>
    <entry>22</entry>
    <entry>23</entry>
    </row>
    <row>
    <entry>31</entry>
    </row>
  </tbody>
  </tgroup>
</table>
```

❶ 说明使用的是何种表格，**table**还是**informaltable**

❷ 表格标题

❸ 定义表格宽度（有多少列），必需！

❹ 表头，表格的第一行，用粗体显示，非必需

❺ 表底，其实我从来没见过有人用它

❻ 表体，表格的主体部分

❼ 行，每一行可以拥有的单元格不能超过表格宽度的定义

❽ 单元格。**entry**元素的内容从该行第一格开始排列

❾ 第一格留空的话，要用 `<entry></entry>` 占一个位置

❿ 留空的单元格之后没有 有内容的单元格，可以省略 `<entry></entry>`

跨行表格

表 **27.2.** 跨行表格

跨两行	12	13
22	23	
31	跨三行	33
41	跨两行	
51		

```
<table>
  <title>跨行表格</title>
  <tgroup cols="3">
    <tbody>
      <row>
        <entry morerows="1">跨两行</entry><entry>12</entry><entry>13</entry>
      </row>
      <row>
        <entry>22</entry><entry>23</entry>
      </row>
      <row>
        <entry>31</entry><entry morerows="2">跨三行</entry><entry>33</entry>
      </row>
      <row>
        <entry>41</entry><entry morerows="1">跨两行</entry>
      </row>
      <row>
        <entry>51</entry>
      </row>
    </tbody>
  </tgroup>
</table>
```

❶ 使用 `morerows="N"` 合并下方的N个单元格；

❷ 由于该行第一个单元格已经被合并，所以应在这里出现的**entry**元素取消

❸ 这里有一个**entry**被取消

❹ 这里有两个**entry**被取消

跨列表格

表 **27.3.** 跨列表格

第一个单元格	第三个单元格	第四个单元格	
第一个单元格	第二个单元格		
第一个单元格	第二个单元格	第三个单元格	第四个单元格

```
<table>
  <title>跨列表格</title>
  <tgroup cols="4">
<colspec colnum="1" colname="1"/>
<colspec colnum="2" colname="2"/>
<colspec colnum="3" colname="3"/>
<colspec colnum="4" colname="4"/>
    <tbody>
      <row>
        <entry namest="1" nameend="2">第一个单元格</entry><entry>第三个单元格</entry><entry>第
      </row>
      <row>
        <entry>第一个单元格</entry><entry namest="2" nameend="4">第二个单元格</entry>
      </row>
      <row>
        <entry>第一个单元格</entry><entry>第二个单元格</entry><entry>第三个单元格</entry><entry
      </row>
    </tbody>
  </tgroup>
</table>
```

> ❶ 先给每列取个名
>
> ❷ 列编号
>
> ❸ 列名。可以任意取，但为了清晰，我用编号作为列名
>
> ❹ 跨列单元格的起始列，用列名指定
>
> ❺ 跨列单元格的结束列

链接

内部链接

转到内部链接

```
<sect1 id="target">被链接章节</sect1>
转到<link linkend="target">被链接章节</link>
```

> ❶ 给被链接的部分指定一个 id
>
> ❷ 给**link**元素添加**linkend**属性，指向刚才定义的 id
>
> ❸ **link**元素内容为链接文字

如果想更精细的控制跳转的目标，可以设置锚点

```
这里是一个锚点<anchor id="anchor"/>

点击跳转到<link linkend="anchor">锚点</link>
```

点击跳转到锚点

外部链接

访问linuxtoy.org

```
访问<ulink url="http://linuxtoy.org">linuxtoy.org</ulink>
```

> ❶ 协议不能省略
>
> ❷ 外部地址
>
> ❸ 链接文字

脚注

这是脚注[54]的写法

```
    <para>这是脚注<footnote><para>脚注示例</para></footnote>的写法</para>
```

> ❶ 脚注内容必须为**para**元素或块元素

参考

参考“内部链接”一节

```
参考<xref linkend="docbook-ln"/>
```

> [54] 脚注示例

告示

告示往往比较有用，它的使用很简单：

```
<note>note</note>
<tip>tip</tip>
<caution>caution</caution>
<important>important</important>
<warning>warning</warning>
```

> 注意：note
>
> 提示：tip
>
> 小心：caution
>
> 重要：important
>
> 警告：warning

图形

插入图形的方法



```
<graphic fileref="img/warning.png"/>
<imagedata fileref="img/warning.png"/>
```

> ❶ DocBook DTD 5.0 将统一使用**imagedata**

使用**figure**作为图形的父元素可以进行编号

图 **27.1.**



```
<figure>
  <imagedata fileref="img/warning.png"/>
</figure>
```

文档工程

如果你在作一个比较大的文档，建议每个章节使用一个单独的文件存储，便于管理[55]

前面讲到的实体可以用来作这件事情

有两个文件一个是主文件 `docbook.xml` ，包含文件头和文档信息；另一个文件 `file.xml` 是文档内容，主文件如下：

```
<?xml version='1.0' encoding="UTF-8"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.5/zh_cn"
"http://www.oasis-open.org/docbook/xml/4.5/docbookx.dtd"
[
<!ENTITY file SYSTEM "file.xml">
]>
<article>
    <info></info>
    &file;
</article>
```

> ❶ 定义外部实体
>
> ❷ 通过外部实体引用文档

`file.xml` 文件内容如下：

```
<sect1>
  <para>
      ……
  </para>
</sect1>
```

- 参见“分子元素”一节

如果把引用文件的列表放在外部文件中，则更加容易管理。但是 XML 不允许在 DTD 中引用普通实体[56]，这就要定义参数实体：

```
<?xml version='1.0' encoding="UTF-8"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.5/zh_cn"
"http://www.oasis-open.org/docbook/xml/4.5/docbookx.dtd"
[
<!ENTITY % list SYSTEM "list.xml">
%list;
]>
```

> ❶ 在定义参数实体时，这里多了一个 `%`
>
> ❷ 引用参数实体时使用 `%` 而不是 `&`

`list.xml` 文件内容如下：

```
<!ENTITY file SYSTEM "file.xml">
<!ENTITY file1 SYSTEM "file1.xml">
<!ENTITY file2 SYSTEM "file2.xml">
```

这样就可以在正文引用 `list.xml` 文件中定义的普通实体了

> [55] 即便文档规模不是很大，把内容独立出来也有好处，比如： 每部分单独发布，修改，节省时间
>
> 如果出错便于排查
>
> [56] 在 DTD 中能够定义外部文件作为普通实体，但是只能在正文中引用，不能在 DTD 中引用

DocBook 编辑软件

我使用 Emacs 编辑 DocBook 源文件，使用 `docbook-xml-mode.el` 这个扩展，在 `.emacs` 文件中加入下面配置：

```
;---------- Docbook
(require 'docbook-xml-mode)

(add-hook 'docbook-xml-mode-hook
      (function (lambda ()
                  (setq outline-regexp "<!\\-\\-\\*+")
              (outline-minor-mode)
              (hide-body))))
```

> ❶ 加载 docbook-xml-mode
>
> ❷ 添加 docbook-xml-mode 钩子，运行下面代码
>
> ❸ 将 `<!--*` 识别为大纲标识
>
> ❹ 启动大纲模式作为辅模式
>
> ❺ 隐藏所有内容，只显示主干

关于大纲模式的使用参见Emacs 大纲模式

Emacs 还有一个 `nxml-mode.el` 也可以用来编写 DocBook，它的优点是可以自动完成 DTD 验证，有语法方面的错误能够实时提示，但是对于多文件的工程支持不够好；而且它插入 XML 标记是通过自动补全，不如 `docbook-xml-mode.el` 方便，DTD 验证和语法检查其实可以由 xsltproc 完成，所以我不用它。

还有一些工具可以生成 docbook-xml 的源文件，如 Emacs-muse，但毕竟不够灵活；通过语法简单的 muse 源文件生成语法复杂的 docbook-xml，所能拥有的特性不会超出 muse 的表现范围，不能够利用 DocBook 强大的能力

发布

使用xsltproc将它发布为 Html

```
xsltproc  /usr/share/xml/docbook/xsl-stylesheets-1.73.2/html/html.xsl docbook.xml
```

> ❶ xsl-stylesheets目录，里面包含可以发布的格式。该路径可能会因系统的不同而改变
>
> ❷ 发布为 Html 格式所需要的 xsl 文件
>
> ❸ 发布为单独的 html 页面。如果需要分页，使用`chunk.xsl`

这是最简单的发布。有的时候，我们需要进行一些控制，例如使用 CSS、设定输出目录等，可以在命令中加入参数：

```
xsltproc --output ../html/                \
--stringparam html.stylesheet docbook.css \
/usr/share/xml/docbook/xsl-stylesheets-1.73.2/html/html.xsl  \
docbook.xml
```

> ❶ 设定输出目录
>
> ❷ 使用 CSS 样式表
>
> ❸ 使用 XSL 样式表

这个命令有点长，我们可以把相关参数写入参数样式表`param.xsl`，使用命令**xsltproc param.xsl docbook.xml**发布

下面是一个参数样式表的例子，复制保存：

例 **27.1. DocBook** 参数样式表

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">

<!--调用样式表-->
<xsl:import href="/usr/share/xml/docbook/xsl-stylesheets-1.73.2/html/chunk.xsl"/>

<xsl:param name="chunker.output.encoding" select="'utf-8'"/>
<!--标准信息所使用的语言-->
<xsl:param name="l10n.gentext.language" select="'zh_cn'"/>
<!--指定样式表-->
<xsl:param name="html.stylesheet" select="'docbook.css'"/>
<!--对于警告类信息是否使用图形 0 1-->
<xsl:param name="admon.graphics" select="1"/>

<!--生成的 HTML 文件存放的起始目录-->
<!--如果没有在 Makefile 或命令中指定，取消这里的注释
<xsl:param name="base.dir" select="'../html/'"/>
-->

<!--生成的 HTML 文件内容是否进行缩排 yes no-->
<xsl:param name="chunker.output.indent" select="'yes'"/>
<!--给节编号 0 1-->
<xsl:param name="section.autolabel" select="0"/>
<!--节的编号是否包含章的编号 0 1-->
<xsl:param name="section.label.includes.component.label" select="1"/>
<!--表格边框的属性是否使用预设 CSS 来指定-->
<xsl:param name="table.borders.with.css" select="0"/>
<!--参考书目是否进行编号-->
<xsl:param name="bibliography.numbered" select="1"></xsl:param>

<!--目录深度-->
<xsl:param name="toc.max.depth" select="2"/>
<!--sect#页面上显示目录-->
<xsl:param name="generate.section.toc.level" select="0"/>
<!--sect#可以生成目录条目-->
<xsl:param name="toc.section.depth" select="2"/>
<!--目录中收录哪些内容-->
<!--包括 toc,title,figure,table,example,equation -->
<!-- nop 为空 -->
<xsl:param name="generate.toc">
appendix  toc
article/appendix  nop
article   toc,title
book      toc,title,example
chapter   toc,title
part      toc,title
preface   toc,title
qandadiv  toc
qandaset  toc
reference toc,title
sect1     toc
sect2     toc
sect3     toc
sect4     toc
sect5     toc
section   toc
set       toc,title
</xsl:param>

<!--在源码中插入 <?linebreak?> 标记，生成 Html 时替换为<br> -->
<xsl:template match="processing-instruction('linebreak')">
<br/>
</xsl:template>

</xsl:stylesheet>
```

将偷懒进行到底，使用下面 `Makefile` ，在工作目录下键入**make**就可以了[57]

```
OBJECT = all
SOURCE =  docbook.xml
PARAM = param.xsl
ARG = --output html/
COMPILER = xsltproc

$(OBJECT):$(SOURCE) $(PARAM)
    $(COMPILER) $(ARG) $(PARAM) $(SOURCE)

clean:
    rm -rf ../html/*.html
```

> ❶ 这里一定要用 TAB，而不能用空格字符
>
> ❷ 同上

> [57] 在 Emacs 中 **M-x compile** 效果更好，如果有错误还能够直接定位

使用 CSS 定制外观

DocBook 输出为 Html 时，如果不使用 CSS 控制，那么它的外观将比较“朴素”

如果自己写一个 CSS 未免太麻烦，可以随便找一个 DocBook 写的文档，将里面的 CSS 文件拿来修改[58]

通过例子简单介绍下 CSS

```
body {
  font-family: verdana, tahoma, helvetica, arial, sans-serif;
  text-align: left;
  background: #fff;
  color: #222;
  margin: 1em;
  padding: 0;
  font-size: 1em;
  line-height: 1.2em
}
```

> ❶ 选择符，**body**标记中的内容如果没有专门指定，都应用花括号中定义的样式
>
> ❷ 花括号中的内容为样式定义
>
> ❸ 字体
>
> ❹ 文字左对齐
>
> ❺ 背景色
>
> ❻ 文字色
>
> ❼ 页边距
>
> ❽ 内部页边距
>
> ❾ 行高
>
> ❿ 行距

- **em**是相对尺寸，当前字体的尺寸=1em；**px**是绝对尺寸，像素，由显示器分辨率决定；字体默认为 16px 大小，也就是说 1em=16px

看看我们生成的 Html 文件的源码

```
<div class="navheader">
<div class="navfooter">
```

> ❶ 给一个元素指定一个类

这里的页眉和页脚，使用的都是**div**标记，只不过类不同，怎么样为它们分别指定样式呢？在 CSS 文件中找到相关段落：

```
div`.`navheader {
  border-bottom: 1px solid #dbddec;
}

div.navfooter {
  border-top: 1px solid #dbddec;
}
```

> ❶ 类选择符，表示类为**navheader**的**div**标记，应用花括号中的样式

如果想对包含某一类的所有标记定义样式，可以单独使用类选择符

```
.command {
color: red;
}
```

- 不管**command**这个类出现在什么标记中，它都应用该样式

如果需要对多个不同的标记定义同一个样式，使用分组选择符

```
div.navheader`,` div.navfooter {
  background: #ecedef;
  margin: 0;
  padding: 0.1em .5em;
}
```

> ❶ 分组选择符

空格为包含选择符

```
table tr td {
border:dashed #999;
  text-align: left;
}
```

> ❶ 表示**table**标记中的**tr**标记中的**rd**标记

代码块的样式

我在 DocBook 中使用 **screen** 输出代码块，看看如何定义它的样式

```
.screen {
  color: #000;
  background-color: #e9e9e9;
  font-weight: normal;
  border: 1px dotted #666699;
  max-height:20em;
  overflow:auto;
}
```

> ❶ 定义边框粗细、类型、颜色。[59]
>
> ❷ 定义最大高度
>
> ❸ 代码块超出上面定义的最大高度时，加入一个滚动条

简单表格的样式

我用简单表格来放置一些需要排列整齐的文字

```
.informaltable table {
 border:0;
 border-top:dashed #999;
 border-width:1px 1px 1px 1px;
 margin-left: 10px;
 margin-right: auto;
}

.informaltable table th, .informaltable table tr td {
 border-right:0;
 border-bottom:dashed #999;
 border-width: 0 1px 1px 0;
 padding: 0.2em 0.4em 0.2em 0.8em;
 text-align: left;
}
```

> ❶ 包含选择符，类为 `informaltable` 的(div)标记，包含的 `table` 标记，应用下面的样式
>
> ❷ **table**标记设定的边框为表格顶部和左边的边框，取消
>
> ❸ 设定顶部的边框为点划线
>
> ❹ 分组选择符，分别选中表头和表体的单元格
>
> ❺ 表格底部和右边的边框由单元格设定[60]
>
> ❻ 文字对齐

> [58] 这个东西你不想要的时候满地都是，一旦你想找一个，可能还真找不到，没关系，可以用我的改。我也是改别人的：）
>
> [59] 边框类型包括：solid实线、dashed点划线、dotted虚线、double双线、groove槽状、ridge脊状、inset凹陷、outset凸出
>
> [60] 确实很别扭，但这不关 DocBook 的事，别忘了我们是通过 CSS 对 Html 进行定义

技巧

关于表格

在目录页中，会用每个表格的标题来生成表格清单。不希望被收录的可以使用 informaltable

可以通过在 CSS 中指定 table 和 informaltable 的样式来定义两种表格

交叉引用

可以放心的先引用，然后再定义被引用的部分。如果被引用的部分没有被定义，发布的时候 xsltproc 会给出提示

calloutlist 自动编号问题

callout 如果不是定义在 screen、programlisting、synopsis 等块元素中，则下一个 calloutlist 中编号并不会重新开始，而是继续上次的。

经测试，callout 并不需要定义在 screen 等元素中，只要两个 calloutlist 中间有这些元素隔开就可以了。

可以定义一个不常用的元素，如 synopsis，来进行分隔。

只要在两个 calloutlist 之间的任何位置插入一个空标记 <synopsis/> 就可以了

当然了，生成的 Html 效果有一点怪异。在 CSS 中加入下面代码可以隐藏。

```
.synopsis {display:none;}
```

尽量把这个标记放在不影响显示的地方，例如两个 section 之间；这样即便意外的显示，也只显示在目录中

断行符

在 param.xsl 文件中加入

```
<xsl:template match="processing-instruction('linebreak')">
<br/>
</xsl:template>
```

在源文件中加入

```
<?linebreak?>
```

生成 Html 时替换为
 这里是演示

内部链接

内部链接时，要给结构元素定义 ID，而不要给 title 定义 ID，不然跳转时会到达错误的位置

第 28 章 Git 版本控制系统

目录

gitweb

版本控制之道

几乎所有项目[61]，都要使用版本控制，它究竟有什么优势呢？

时间机器

假设你使用的编辑器，不支持删除，那你就得特别的谨小慎微，甚至是如履薄冰：因为你打错了字没法删除

放松下来，目前我所接触的所有编译器中，还没有变态到这种程度的。

如果编译器提供了删除功能，却没有 undo，那可能会更可怕：如果你不小心选中了全部文字，手一抖……因为不能 undo，你知道，如果此时不小心按下 delete，你就得从头来过……你会为可能产生的后果而发抖

然而，命运总在你不想被打扰的时候来敲你可爱的门，在你手抖的刹那，你真的鬼使神差的按了下那个可怕的键……你的选择只能是重新来过或者放弃……

你会比任何时候都希望时间倒流一秒钟

还好，几乎所有的编译器，都会帮你回到一秒种、一分钟、一小时……之前

假设你想回到一天之前？一觉醒来，你突然想起，有一部分内容其实应该保留下来……但是编辑器在重启之后，就不能够再帮你回到以前的任何时间……

这种情况下，版本控制才是你的救命稻草

分支控制

如果项目只能朝一个方向发展，你会时常在确定方向的问题上犹豫不决。而使用版本控制，创建一个分支各自发展，在适当的时候合并分支，是最好的解决办法

协同工作

很多项目需要协同工作，版本控制能够提供协同工作需要的环境，解决协同工作可能产生的问题

冲突合并

项目成员可能会对一处内容进行不同的修改，版本控制能够反馈这些冲突，以便解决它

常见版本控制系统

传统的版本控制系统，需要在服务器上搭建一个中央仓库，所有成员都是客户端，具有不同的权限。这种方式适应大教堂开发模式，但比较依赖网络，且不够灵活，使用比较广泛的有cvs、svn等

而分布式版本控制系统，则不需要中央仓库，所有的成员都可以完全掌控己方的版本控制系统，通过多种方式灵活的协作。这种方式适应集市开发模式，典型的代表是git

---

> [61] 项目并不总是意味着开发一个大型软件；你帮领导打一份文件，其实也是项目

git 如何工作

尽管 `Linus Torvalds` 将 git 定位为："傻瓜式的内容跟踪工具"，但它对不熟悉版本控制的朋友来说，还是过于复杂

所以我们需要先在概念上大概了解，git 是如何工作的

补丁

多数版本控制系统，使用补丁来纪录内容的改动。

当你修改了文件内容，版本控制系统会比较修改后的内容和原来的内容，并使用补丁纪录下来。 无论是查看版本之间的变化，或者需要回溯原来内容，都需要使用补丁中的内容

git 对象

工作树

git将工作目录称为：工作树

索引

工作树的快照，无论是添加、删除文件，或者对文件内容进行修改，都需要提交到索引。git只跟踪被索引的内容

将改动提交到索引，意味着建立一个快照

版本库

存储工作树的各种版本

工作树中只保存当前内容，各种版本通过补丁的形式存储在版本库中

版本名称

git可以使用“版本ID”和“版本标签”作为版本名称

版本ID自动生成，版本标签用**git tag**命令指定

操作级别

git可以在四种级别上实现版本控制：

改动纪录

改动了文件内容，提交到索引，但未提交到版本库

该级别的常见操作有：**add diff**

版本纪录

改动被提交到版本库后，就成为一个新的版本

该级别的常见操作有：**commit log tag show reset**

其中**reset**、以及分支操作，需要在**commit**之后，**add**之前，没有待提交改动纪录的情况下进行

分支

分支为该主线上的系列版本

版本库

协同工作时，需要合并项目成员的版本库

该级别常见的操作有：**clone pull push**

初始化

创建版本库

git 基于文件夹(工作树)进行版本控制，在一个文件夹中创建 git版本库：

```
$ cd project/
$ git init
Initialized empty Git repository in .git/
```

> ❶ 输出信息：在当前文件夹的 `.git/` 目录下创建版本库

将文件提交到 git索引：

```
git add file1 file2 file3 ……
```

更方便的作法是将当前文件夹中的所有文件全部加入到索引中

```
git add .
```

- 可以在 `.gitignore` 文件中设置排除的文件(通常把临时文件排除)

> 注意：git 只负责管理被索引的文件

此时，文件还没有被提交到版本库。向版本库提交第一个版本：

```
git commit
git commit -m "备注"
```

> ❶ 调用系统默认编辑器编辑备注内容

版本库状态

使用 **git status** 命令查看版本库状态。先创建一个演示版本库：

```
mkdir sandbox                  #新建一个文件夹
cd sandbox/                    #进入该文件夹
git init                       #初始化版本库
touch a b                      #新建 a b 两个文件
git add .                      #将这两个文件提交到索引
git commit -m "创建git版本库"  #将第一个版本提交到版本库
```

这时使用 `git status` 查看版本库状态：

```
# On branch master
nothing to commit (working directory clean)
```

对文件进行一些操作：

```
vi a      #编辑 a
rm b      #删除 b
touch c   #新建 c
```

再用 `git status` 查看：

```
# On branch master           #在 master 分支上
# Changes to be committed:  #已提交到索引，等待提交到版本库(其实本例中没有这一段)
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:  e
#       modified:  f
#
# Changed but not updated:  #改动未提交到索引
#   (use "git add/rm <file>..." to update what will be committed)
#
#       **modified:   a**
#       **deleted:    b**
#
# Untracked files:            #文件未提交到索引
#   (use "git add <file>..." to include in what will be committed)
#
#       **c**
no changes added to commit (use "git add" and/or "git commit -a")
```

> 注意：如果只是想删除该文件夹中的版本库，只要删除 `.git/` 目录即可

```
rm -rf .git
```

配置

git 初始化后，会在 `.git/` 目录下创建一个版本库，其中 `.git/config` 为配置文件。

用户信息

为当前版本库添加用户信息[62]：

```
[user]
    name = kardinal
    email = 2999am@gmail.com
```

也使用全局用户信息，在 `~/.gitconfig` 中写入上述内容，或者使用命令：

```
git config --global user.name "kardinal"
git config --global user.email 2999am@gmail.com
```

语法高亮

在 `~/.gitconfig` 文件中添加如下语句，使用容易阅读的彩色来输出信息：

```
[color]
    branch = auto
    diff = auto
    status = auto
```

或者自己定义：

```
branch.current          # color of the current branch
branch.local            # color of a local branch
branch.plain            # color of other branches
branch.remote           # color of a remote branch
diff                    # when to color diff output
diff.commit             # color of commit headers
diff.frag               # color of hunk headers
diff.meta               # color of metainformation
diff.new                # color of added lines
diff.old                # color of removed lines
diff.plain              # color of context text
diff.whitespace         # color of dubious whitespace
status                  # when to color output of git-status
status.added            # color of added, but not yet committed, files
status.changed          # color of changed, but not yet added in the index, files
status.header           # color of header text
status.untracked        # color of files not currently being tracked
status.updated          # color of updated, but not yet committed, files
```

> [62] 这是必需的，请不要忽略

版本更新

现在创建一个 git版本库：(参见“初始化”一节)

```
mkdir sandbox
cd sandbox/
git init
touch test
git add .
git commit -m "创建git版本库"
```

git log查看版本纪录：

```
commit d63e709f565dcd60ab749f0eca27a947b02b8c26
Author: kardinal <2999am@gmail.com>
Date:   Wed Nov 5 14:08:50 2008 +0800

    创建 git版本库
```

> ❶ 版本ID(默认自动生成)
>
> ❷ 提交者
>
> ❸ 提交日期
>
> ❹ 备注

现在对 `test` 文件作一些修改：

```
增加一行内容
```

git diff查看自上次提交以来发生什么改动：

```
diff --git a/test b/test
index e69de29..bae0882 100644
--- a/test
+++ b/test
@@ -0,0 +1 @@
+增加一行内容
```

> ❶ 典型的diff输出，如果你设置了彩色输出，这些内容会非常直观的显示

接下来，把这次的更新作为新的版本提交

```
git add test
git commit -m "增加了一行内容"
```

> ❶ 将本次更新提交到索引(生成快照)。此时使用**git diff**查看改动纪录，看不到任何内容；但是仍可以使用**git diff --cached**查看缓存的改动纪录
>
> ❷ 提交为新版本后，便不能使用**git diff**查看改动纪录
>
> 提示：`git add`提交改动到索引，但并不提交到版本库。如果不想频繁的提交新版本，可以使用该命令提交改动到索引，比较和上一次提交的变化。只要不使用`git commit`提交，版本库中不会有新的版本

使用**git log**查看版本库纪录

```
commit 13aa16309db3693ea8a6b93b8a818e731194824c
Author: kardinal <2999am@gmail.com>
Date:   Wed Nov 5 14:28:04 2008 +0800

    增加了一行内容

commit d63e709f565dcd60ab749f0eca27a947b02b8c26
Author: kardinal <2999am@gmail.com>
Date:   Wed Nov 5 14:08:50 2008 +0800

    创建git版本库
```

如果想查看每个版本的改动纪录，使用**git log -p**

```
commit 13aa16309db3693ea8a6b93b8a818e731194824c
Author: kardinal <2999am@gmail.com>
Date:   Wed Nov 5 14:28:04 2008 +0800

    增加了一行内容

diff --git a/test b/test
index e69de29..bae0882 100644
--- a/test
+++ b/test
@@ -0,0 +1 @@
+增加一行内容

commit d63e709f565dcd60ab749f0eca27a947b02b8c26
Author: kardinal <2999am@gmail.com>
Date:   Wed Nov 5 14:08:50 2008 +0800

    创建git版本库

diff --git a/test b/test
new file mode 100644
index 0000000..e69de29
```

每次使用**git add**和**git commit**两个命令提交版本更新很繁琐，可以使用**git commit -a**提交(已索引文件的改动)

```
git commit -a -m "一次新的提交"
```

版本标签

使用**git tag**为某一版本创建版本标签：

```
git tag 1.0 d63e70
git tag 1.1 13aa16
git tag newest HEAD
```

- 版本标签存储在 `.git/refs/tags/` 目录

使用容易记忆的版本标签进行操作：

```
git diff 1.0 1.1
git diff 1.0 13aa16
git log 1.0
```

> ❶ 查看1.0和1.1之间的变化
>
> ❷ 查看1.0纪录

时间机器

在 `test` 文件中随意改动，然后提交

```
git commit -a -m "意外改动"
```

`git log`，增加了一条纪录：

```
commit d9b03125921d20482937f43ea0bdbfbfb7fe1745
Author: kardinal <2999am@gmail.com>
Date:   Wed Nov 5 15:18:49 2008 +0800

    意外改动
```

使用**git reset**命令回溯到历史版本：

```
git reset HEAD^
git log
git diff
```

> ❶ `git reset`默认使用**--mixed**选项
>
> ❷ `HEAD`表示当前版本，`HEAD^`表示前一个版本，`HEAD^^`表示前两个版本，`HEAD~4`表示前四个版本；也可以使用“版本标签”或“版本ID”来指定版本(只要前几位就可以了)
>
> ❸ 可以看到版本纪录中最后一次提交已经取消
>
> ❹ 可以看到，**--mixed**选项回溯到提交到索引之前的状态

git reset --soft回溯到已提交到索引但未提交到版本库的状态

```
git commit -a -m "意外改动"
git reset --soft HEAD^
git log
git diff
git diff --cached
```

> ❶ 再一次将这些改变提交
>
> ❷ 使用**--soft**选项回溯到上一版本
>
> ❸ 版本纪录中已取消该版本
>
> ❹ 改动纪录中没有任何内容
>
> ❺ 改动已被提交到索引，但是未提交到版本库，所以缓存的改动纪录还可以查看
>
> 注意：`git reset` 回溯到`git add`之前的状态；`git reset --soft`回溯到`git add`之后的状态

以上方法回溯到历史版本，只是回溯版本库和索引的纪录，而文件的内容并不会回溯到之前的状态，使用**git reset --hard**命令，将文件内容也一同回溯

```
git commit -a -m "意外改动"
git reset --hard HEAD^
git log
git diff --cached
cat test
```

> ❶ ......还得提交一次，谁让它是“意外改动”
>
> ❷ 使用**--hard**选项回溯到上一版本
>
> ❸ 版本纪录中已取消该版本
>
> ❹ 没有任何改动纪录待提交
>
> ❺ 文件内容回溯到上一版本的状态

--hard选项存在一定风险，因为很多情况下，你不能确定内容算不算“意外改动”。这时，可以新建一个分支，在这个分支中进行回溯，处理完成后合并两个分支，参见“分支管理”一节

分支管理

创建分支

git branch命令查看分支：

```
git branch
* master
```

> ❶ 不带选项，默认为查看分支
>
> ❷ `*` 表示当前分支
>
> ❸ `master` 为默认分支

新建分支：

```
$ git branch slave
$ git checkout slave
M       slave
Switched to branch "slave"
$ git branch
  master
* slave
```

> ❶ `git branch`使用分支名称作参数，新建分支
>
> ❷ `git checkout` ，切换到指定分支
>
> ❸ 查看分支
>
> ❹ 当前分支已变为`slave`

使用如下命令删除分支：(先不要删除，后面会用到)

```
git branch -D 分支名称
```

合并分支

使用**git merge**合并分支：

```
编辑 test
git commit -a -m "slave分支"
git checkout master
git diff master slave
git merge slave
```

> ❶ 增加一点内容
>
> ❷ 在当前分支提交此版本
>
> ❸ 切换到 master分支
>
> ❹ 比较两个分支
>
> ❺ 合并 slave分支 的内容

处理冲突

如果没有冲突的内容，git 会自动处理合并。如果产生冲突(同一行的内容不一致)，git 会输出如下信息：

```
Auto-merged test
CONFLICT (content): Merge conflict in test
Automatic merge failed; fix conflicts and then commit the result.
```

- `test` 文件在合并时发生冲突，需要手动处理冲突，然后后再次提交

现在处理冲突，打开 `test` 文件，有如下内容：

```
<<<<<<< HEAD:test
这是master分支中的一行
=======
这是slave分支中的一行
>>>>>>> slave:test
```

> ❶ 当前内容信息
>
> ❷ 当前内容
>
> ❸ 分隔线，分隔冲突的内容
>
> ❹ slave分支内容
>
> ❺ slave分支:test文件

修改这部分内容，保留正确的，然后提交

> 提示：冲突不只在合并分支时产生。无论何种冲突，处理的方法是一样的

合并后可以删除该分支：

```
git brancd -d slave
```

> ❶ **-D**强行删除分支；**-d**只有分支内容被合并后才能删除

通过文件协作

git 可以通过补丁文件进行协作(使用 email 传送补丁文件)

首先通过 **git clone** 创建一个镜像版本库，使用 `git branch -a` 命令查看所有分支

```
$ git clone http://linuxtoy.org/path  [local]
$ cd [local]
$ git branch -a
* master
  origin/HEAD
  origin/master
```

> ❶ 原始版本库路径
>
> ❷ 镜像版本库路径。它是可选的，如果没有指定，则使用和发起者同样的路径(文件夹名称)

其中 `origin` 为原始版本库镜像，在 master 分支上的工作，要生成对于 origin 的补丁，origin 必须与原始版本库保持一致，不要试图修改它

```
git fetch origin   #更新 origin 分支。如果 origin 分支不是最新的原始版本库，会产生错误的补丁文件
git rebase  origin  #将工作迁移到最新原始版本库基础上
git format-patch origin #生成补丁文件
```

- 使用 **git rebase** 后可能会产生冲突，手动处理

生成的补丁文件为 `0001-[备注].patch` ，发起者得到补丁后，使用 **git am** 命令将这个补丁应用到版本库

```
git checkout -b patched
git am 0001-[备注].patch
git checkout master
git diff master patched
git merge patched
```

> ❶ 为谨慎起见，创建一个名为 “patched” 的分支，切换到此分支
>
> ❷ 在 “patched” 分支中应用补丁

通过网络协作

git 提供相当灵活的协作方式，最常见的方式为：协作者获得原始版本库的镜像，并在上面工作；发起者从协作者那里获取更新

协作者通过**git clone**创建一个镜像版本库：

```
git clone user@url:~/path [local]
```

网络对于 git 来说是透明的，凡是可以访问的位置，如 http、ftp、ssh……，甚至本地路径，对于 git 来说没有什么区别。

通过以下命令，创建一个本机原始版本库 `sandbox` 的镜像 `project` ，是允许的：

```
git clone ~/sandbox project
```

对于没有指定协议的远程路径，git 默认使用 ssh

```
(ssh://)user`@`127.0.0.1`:`~/sandbox
```

使用**git pull**获取协作者版本库中的内容：

```
git pull user@127.0.0.1:~/sanbox master[:newest]
```

> ❶ 版本库
>
> ❷ 分支名称
>
> ❸ 版本名称(可选。使用版本ID、版本标签，请不要使用“HEAD”)
>
> 提示：**git pull** 基于“版本”操作，也就是说，只有提交后才可以进行；这个命令会比较两个版本的时间戳，只获取更新的版本

当发起者进行了更新后，协作者应从发起者那里获取最新的原始版本库，并将当前工作迁移到最新的原始版本库基础上

```
git fetch origin          #获取最新原始版本库
git rebase origin/master #将工作迁移到最新原始版本库
```

这时发起者再次使用 **git pull** 从协作者那里获取更新……

gitweb

首先配置 web 服务器，使其支持 cgi，参见“CGI”一节

将 git 工作树拷贝到 web 服务器目录下:

```
cp -r sandbox /home/lighttpd/html/
```

gitweb 通常随 git 一同安装，拷贝文件到 git 工作树

```
cp /usr/share/gitweb/* /home/lighttpd/html/sandbox
```

检查 `/home/lighttpd/html/sandbox/gitweb.cgi` 文件中的如下语句

```
our $GIT = "/usr/bin/git";
our $projectroot = ".";
```

> ❶ git 执行文件位置
>
> ❷ 项目根目录，也可以使用绝对路径，如 `/home/lighttpd/html/sandbox`

修改项目描述，编辑项目根目录下的 `.git/description` 文件

这样就建立了一个 gitweb 站点，通过以下地址访问：

```
http://linuxtoy.org/sandbox/gitweb.cgi
```

如果想通过 http 协议使用，例如：

```
git clone http://linuxtoy.org/sandbox/.git
```

则需要在项目根目录下执行 **git update-server-info**

第 29 章 ConTeXt 入门指南

目录

TeX 简介

古来圣贤皆寂寞，唯有饮者留其名

沧海桑田，UNIX 的世界在风雨飘摇中悄悄变换了容颜

从传统工具到 GNU 工具，从 4.4BSD-Lite 到 FreeBSD，从 Minix 到 Linux，从 VI 到 VIM……往日的 UNIX 已经不复存在

唯一历久弥新的，只有 UNIX 的风骨；或者还有，TeX？

为什么要用 TeX

TeX 足够强大，可以满足使用者的绝大多数要求。尤其在排版质量、数学公式、图形生成、结构化控制等方面，几乎无出其右者

尽管学习 TeX 相对困难，但是学习的收益比成本要大得多

TeX 尤其适合需要打印的文档。恐怕你不能够在办公室里使用诸如 DocBook、reStructuredText 之类的格式写一份文件，然后把它拿给你的老板

顺流、逆流

现代文档工具，倾向于分离内容与样式。而内容部分，则用结构化的方式进行梳理，对于规模比较大的文档，这是最佳的方案。

Knuth 的 TeX 本身没什么结构控制，它就像一套排版领域的"汇编语言"，大约 300 多个标记。TeX 在处理一份文档时，就是把整篇文档当作一个很长很长的字符串一口吞下去，然后消化掉，最后排出 dvi 文件（现代的 TEX 输出的是 PDF）。

由于 TeX 支持宏扩展，为了方便排版，Knuth 做了一个 Plain TeX，对 TeX 标记进行了一些功能逻辑上的封装。不过，还是很不结构化。

LaTeX 是较早尝试对 TeX 进行结构化封装的宏包，它可以明确地实现样式与内容的分离。不过，LaTeX 本身对排版方面的内建支持太少，需要许多其它宏包的支持，而那些宏包不是统一开发的，它们之间经常出现冲突。

ConTeXt 是对 TeX 进行结构化封装的新尝试，比 LaTeX 更加结构化，对排版方面的内建支持很好。而且 ConTeXt 对 XML 的支持相对完善，甚至可以直接用 ConTeXt 的 XML 语法来写文档[63]。

总体说来，LaTex 是科技论文事实上的标准，如果要排论文，就用 LaTeX；而作个人的文档、非正式散布的文档、演示文档，推荐用 ConTeXt

---

> [63] DocBook与ConTeXt之间可以无缝转换

LaTeX

部署 LaTeX

安装 texlive-core

新建 `a.tex` 文件，内容如下：

```
\documentclass[11pt,a4paper]{article}

%加入了一些针对XeTeX的改进并且加入了 \XeTeX 命令来输入漂亮的XeTeX logo
\usepackage{xltxtra}
%启用一些LaTeX中的功能
\usepackage{xunicode}

%%%% fontspec 宏包 %%%%
\usepackage{fontspec}
% 指定字体
\setmainfont[BoldFont=Adobe Heiti Std]{Adobe Song Std}
\setsansfont[BoldFont=Adobe Heiti Std]{Adobe Kaiti Std}
%\setmonofont{Bitstream Vera Sans Mono}

%%%% 版面 %%%%
\usepackage[top=1in,bottom=1in,left=1.25in,right=1in]{geometry}
% 设置行距
\linespread{1.3}

%%%% 缩进 %%%%
% 自动首行缩进
\usepackage{indentfirst}
% 设置首行缩进的距离
\setlength{\parindent}{2.22em}

%连字符
\defaultfontfeatures{Mapping=tex-text}
%中文断行
\XeTeXlinebreaklocale "zh"
\XeTeXlinebreakskip = 0pt plus 1pt minus 0.1pt

%以下为正文部分
\begin{document}
% 生成目录
\tableofcontents
% 章节
\section{\XeLaTeX}
XeTeX 可以使用系统自带的字体，而不需要再另外生成
% 章节
\section{……}
\end{document}
```

使用 `fc-list` 命令查看可用字体，根据实际情况修改 % 指定字体 下面的几行

使用以下命令生成 PDF

```
xelatex a.tex
```

安装宏包

LaTeX 只是一个基本框架，通过宏包扩展其功能。例如：

fontspec

指定字体

geometry

设置版面

color

使用色彩

hyperref

设置链接

listings

代码引用环境

titlesec

设置标题样式

titletoc

设置目录样式

indentfirst

首行缩进(这是中文的排版习惯)

xltxtra

XeTeX 扩展功能

到 CTAN 下载宏包，解压后移动到 LaTeX 能找到的目录，使用以下命令刷新：

```
sudo texhash
```

学习 LaTeX

推荐两本 LaTeX 的书：

- lshort-zh-cn 4.20
- lnotes

Latex 结构化写作

对于规模比较大的文档项目，要用结构化方法写作

首先样式要与内容分离，将 LaTeX 的导言部分写在一个文件里，例如 `style.tex`：

```
%\documentclass[11pt,a4paper,twoside]{article}
\documentclass[11pt,a4paper,twoside]{book}

%加入了一些针对XeTeX的改进并且加入了 \XeTeX 命令来输入漂亮的XeTeX logo
\usepackage{xltxtra}
%启用一些LaTeX中的功能
\usepackage{xunicode}

%%%% fontspec 宏包 %%%%
\usepackage{fontspec}
% 指定字体
\setmainfont[BoldFont=Adobe Heiti Std]{Adobe Song Std}
\setsansfont[BoldFont=Adobe Heiti Std]{Adobe Kaiti Std}
%\setmonofont{Bitstream Vera Sans Mono}

%%%% 版面 %%%%
\usepackage[top=1in,bottom=1in,left=1.25in,right=1in]{geometry}
% 设置行距
\linespread{1.3}

%%%% 缩进 %%%%
% 自动首行缩进
\usepackage{indentfirst}
% 设置首行缩进的距离
\setlength{\parindent}{2.22em}

%连字符
\defaultfontfeatures{Mapping=tex-text}
%中文断行
\XeTeXlinebreaklocale "zh"
\XeTeXlinebreakskip = 0pt plus 1pt minus 0.1pt

%%%% color %%%%
\usepackage{color}
\definecolor{gray}{rgb}{0.9,0.9,0.9}
\definecolor{blue}{rgb}{0,0,1}

%%%% 章节标题 %%%%
\usepackage[center,pagestyles]{titlesec}
\titleformat{\section}{\centering\Large\bfseries}{\thesection}{1em}{}
\titleformat{\subsection}{\large\bfseries}{\thesubsection}{1em}{}

\newpagestyle{main}{
\sethead{\small\thesection\quad\sectiontitle}{}{$\cdot$~\thepage~$\cdot$}
\setfoot{}{}{}\headrule}
\pagestyle{main}

%%%% 目录样式 %%%%
\usepackage{titletoc}
\titlecontents{chapter}
              [0.0em]
              {}  %\song
              {\thecontentslabel\hspace{1em}}
```

```
                {}
                {\normalfont\dotfill\textrm{\contentspage[{\bfseries\thecontentspage}]}}
\titlecontents{section}
                [0.0em]
                {}  %\song
                {\thecontentslabel\hspace{1em}}
                {}
                {\normalfont\dotfill\textrm{\contentspage[{\bfseries\thecontentspage}]}}

%%%% hyperref %%%%
\usepackage[
pdfstartview=FitH, CJKbookmarks=true, bookmarksnumbered=true,
bookmarksopen=true,
linkcolor=true, %注释掉此项则交叉引用为彩色边框(将colorlinks和pdfborder同时注释掉)
colorlinks=blue,
pdfborder=001, %注释掉此项则交叉引用为彩色边框
citecolor=blue ]{hyperref}

\usepackage{listings}
    \lstset{
        numbers=none,
        numberstyle=\scriptsize,
        frame=single,framerule=0.1pt,
        backgroundcolor=\color{gray},
        fontadjust=false,
        flexiblecolumns=true,
        language=[LaTeX]TeX,
        basicstyle=\ttfamily\small,
        commentstyle=\color{orange},
        keywordstyle=\color{blue}
escapebegin=\begin{esc},escapeend=\end{esc},texcl=true
        }

\graphicspath{{img/}}

%其它
%\usepackage[marginal,perpage,symbol]{footmisc}
```

将所有会用的到词汇在一个文件中定义，例如 `glossary.tex`

```
\def\xxx{The five boxing wizards jump quickly.\\}
```

- 使用命令 **\xxx** 插入定义的词汇

使用 **\input** 命令将这两个文件插入到主文档中：

```
%插入样式定义文件的内容
\input{style}
%插入词汇定义文件的内容
\input{glossary}

%正文内容
\begin{document}

%将前言放到 `info.tex` 文件中
%使用 \include 命令载入
\include{info}

%插入目录
\tableofcontents

%将每章的内容放在单独的文件中 (1st.tex)
%使用 \include 命令载入
% `1st.tex` 文件中应包含 **\chapter** 等命令
\include{1st}
% \include 命令会在新的一页上排版载入的文本
%如果不想分页，可以使用 \input 命令，它只是简单的载入文本 (2nd.tex)
\input{2nd}
\end{document}
```

ConTeXt 简介

LaTeX 是一个基本框架，主要依靠宏包扩展功能，这样在保持自身精干的同时，可以拥有强大的功能，不失为一个灵活的方案

这种方案初衷是好的，但是在漫长的进化中，零零散散开发的宏包缺乏组织，多个宏包可能同时提供某一功能，这就可能引发冲突……而 ConTeXt 则不存在这一问题，因为它是的各种功能是统一开发的，而且开发进程相当活跃。

ConTeXt 制作的 PDF 文档，可以拥有强大的交互特性，使用 ConTeXt 作出的 PDF 文档，甚至可以作为计算器

当然，ConTeXt 并不是很容易掌握的，千里之行，始于足下，我们先搭建 ConTeXt 的工作环境

部署 ConTeXt

安装

目前，只有 ConTeXt Minimals （ConTEXt最小包）提供了最新的 ConTeXt 版本，而且可以与系统内其它 TeX 发行版和睦相处

ConTeXt Minimals 通过脚本安装，安装脚本会下载安装文件到当前目录。为了避免混乱，有必要手动为 ConTeXt Minimals 创建目录

```
export CTXDIR=/opt/context
mkdir -p $CTXDIR
cd $CTXDIR
```

> ❶ 这是推荐的目录

使用 rsync 下载 ConTeXt Minimals 的安装脚本：

```
rsync -ptv rsync://contextgarden.net/minimals/setup/linux/first-setup.sh .
```

- 同样，安装脚本也使用 rsync 下载安装文件。这种方式只下载升级过的文件，便于增量更新；也能够最大程度的降低网络不稳定所带来的影响

执行安装脚本64

```
./first-setup.sh
```

- 升级到最新版本，也使用这个脚本

下载完所有文件后，安装脚本会自动配置 ConTeXt Minimals [65]

快速通道

新建一个 `ctx` 文件，内容如下：

```
export CTEXDIR=/opt/context
source $CTEXDIR/tex/setuptex $CTEXDIR/tex
export OSFONTDIR="/usr/share/fonts/adobe"
```

> ❶ 这里的目录为 ConTeXt Minimals 安装目录
>
> ❷ ConTeXt 需要设置一些环境变量才能正常工作，使用 `setuptex` 脚本
>
> ❸ `OSFONTDIR` 为字体路径变量，设置为系统中字体的实际路径，见“字体设置”一节

执行 `source ctx` 命令，设置好需要的环境变量，ConTeXt 工作环境便准备就绪

新建一个 `a.tex` 文件，内容如下：

```
% a.tex      %这里是注释
\starttext   %正文开始
Hello World. %正文内容
\stoptext    %正文结束
```

执行 `context a.tex` 命令，会得到 `a.pdf` 文件

> 提示：执行 `context --purge` 命令清除中间文件。但保留中间文件可以提高编译速度

字体设置

下面是一个包含中文的 `chinese.tex` 文件

```
\usetypescriptfile[zhfonts]     %加载打印字体配置文件(typescript) `zhfonts.tex`
\usetypescript[myscript]        %使用打印字体配置文件中定义打印字体的脚本 `myscript`
\setupbodyfont[myfont,rm,11pt]  %设置正文字体

\setupindenting[always,2em,first]                    %设置中文缩进格式(首行缩进两字)
\setupheads[indentnext=yes]                          %每节的首段也要缩进
\setupinterlinespace[big]                            %设置行距(big=1.5倍)
\setupwhitespace[medium]                             %设置段间距[small, medium, big]
\definepapersize[SCREEN][width=21cm,height=29.7cm] %设置页面尺寸
\setuppapersize[SCREEN][SCREEN]                      %纸张尺寸，通常和页面尺寸相同。(除非在印刷用纟

\starttext
世界，你好！
\stoptext
```

`打印字体配置文件` (typescript) 文件是一个 TeX 文件，要求与 `chinese.tex` 文件在同一目录，或者位于 ConTeXt Minimals 可以检索到的某个目录中[66]

新建一个 `打印字体配置文件` (typescript)，名为 `zhfonts.tex` ，内容如下：

```
% engine=luatex
% \ctxlua{fonts.collections.trace = true}
\let\synchronizetext\relax
\synchronizemathfontsfalse

\spaceskip .25em plus .25em \relax

\definefontfeature[zh][mode=node,script=hang,lang=zhs]

\starttypescript [serif] [zhfont]
    \definefontsynonym
    [ZhSerif][name:AdobeSongStd-Light][features=zh]
    \definefontsynonym
    [ZhSerifBold][name:AdobeHeitiStd-Regular][features=zh]
    \definefontsynonym
    [ZhSerifItalic][name:AdobeKaitiStd-Regular][features=zh]
    \definefontsynonym
    [ZhSerifBoldItalic][name:AdobeHeitiStd-Regular][features=zh]

    \definefontfallback
    [WesternSerif][name:TeXGyrePagella-Regular][0x0000-0x0400][force=yes]
    \definefontfallback
    [WesternSerifBold][name:TeXGyrePagella-Bold][0x0000-0x0400][force=yes]
    \definefontfallback
    [WesternSerifItalic][name:TeXGyrePagella-Italic][0x0000-0x0400][force=yes]
    \definefontfallback
    [WesternSerifBoldItalic][name:TeXGyrePagella-BoldItalic][0x0000-0x0400][force=yes]

   \definefontsynonym
    [Serif][ZhSerif][fallbacks=WesternSerif]
    \definefontsynonym
    [SerifBold][ZhSerifBold][fallbacks=WesternSerifBold]
    \definefontsynonym
    [SerifItalic][ZhSerifItalic][fallbacks=WesternSerifItalic]
    \definefontsynonym
    [SerifBoldItalic][ZhSerifBoldItalic][fallbacks=WesternSerifBoldItalic]
\stoptypescript

\starttypescript [sans][zhfont]
```

```
    \definefontsynonym
    [ZhSans][name:AdobeKaitiStd-Regular][features=zh]
    \definefontsynonym
    [ZhSansBold][name:AdobeHeitiStd-Regular][features=zh]
    \definefontsynonym
    [ZhSansItalic][name:AdobeKaitiStd-Regular][features=zh]
    \definefontsynonym
    [ZhSansBoldItalic][name:AdobeHeitiStd-Regular][features=zh]

    \definefontfallback
    [WesternSans][name:TeXGyreAdventor-Regular][0x0000-0x0400][force=yes]
    \definefontfallback
    [WesternSansBold][name:TeXGyreAdventor-Bold][0x0000-0x0400][force=yes]
    \definefontfallback
    [WesternSansItalic][name:TeXGyreAdventor-Italic][0x0000-0x0400][force=yes]
    \definefontfallback
    [WesternSansBoldItalic][name:TeXGyreAdventor-BoldItalic][0x0000-0x0400][force=yes]

    \definefontsynonym
    [Sans][ZhSans][fallbacks=WesternSans]
    \definefontsynonym
    [SansBold][ZhSansBold][fallbacks=WesternSansBold]
    \definefontsynonym
    [SansItalic][ZhSansItalic][fallbacks=WesternSansItalic]
    \definefontsynonym
    [SansBoldItalic][ZhSansBoldItalic][fallbacks=WesternSansBoldItalic]
\stoptypescript

\starttypescript [mono][zhfont]
    \definefontsynonym
    [ZhMono][name:AdobeFangsongStd-Regular][features=zh]
    \definefontsynonym
    [ZhMonoBold][name:AdobeHeitiStd-Regular][features=zh]
    \definefontsynonym
    [ZhMonoItalic][name:AdobeFangsongStd-Regular][features=zh]
    \definefontsynonym
    [ZhMonoBoldItalic][name:AdobeHeitiStd-Regular][features=zh]

    \definefontfallback
    [WesternMono][name:TeXGyreCursor-Regular][0x0000-0x0400][force=yes]
    \definefontfallback
    [WesternMonoBold][name:TeXGyreCursor-Bold][0x0000-0x0400][force=yes]
    \definefontfallback
    [WesternMonoItalic][name:TeXGyreCursor-Italic][0x0000-0x0400][force=yes]
    \definefontfallback
    [WesternMonoBoldItalic][name:TeXGyreCursor-BoldItalic][0x0000-0x0400][force=yes]

    \definefontsynonym
    [Mono][ZhMono][fallbacks=WesternMono]
    \definefontsynonym
    [MonoBold][ZhMonoBold][fallbacks=WesternMonoBold]
    \definefontsynonym
    [MonoItalic][ZhMonoItalic][fallbacks=WesternMonoItalic]
    \definefontsynonym
    [MonoBoldItalic][ZhMonoBoldItalic][fallbacks=WesternMonoBoldItalic]
\stoptypescript

\starttypescript[myscript]
    \definetypeface[myfont][rm][serif][zhfont]
    \definetypeface[myfont][ss][sans][zhfont]
    \definetypeface[myfont][tt][mono][zhfont]
\stoptypescript
```

- 如何定制字体参阅“打印字体配置文件”一节

下载中文字体[67]，安装，并设置 `OSFONTDIR` 变量：

```
export OSFONTDIR="/usr/share/fonts/adobe"
```

> ❶ 中文字体的安装路径，建议写入 `ctx` 文件[68]

执行以下命令刷新文档数据库：

```
context --generate
```

最后，使用 `chinese.tex` 文件生成 PDF

```
  context chinese.tex
```

> [64] 需要 ruby 支持
>
> [65] 期间可能会出现几次如下> 提示：

```
! I can't find file `core-swd'.
```

键入 `core-swd.mkii` 即可

> [66] 如 `$TEXMFLOCAL``/tex/context/third`

```
mkdir -p $TEXMFLOCAL/tex/context/third
mv zhfonts.tex $TEXMFLOCAL/tex/context/third
context --generate
```

> ❶ 在 `$TEXMFLOCAL` 目录下新建第三方目录
>
> ❷ 将 `zhfonts.tex` 文件移动到该目录
>
> ❸ 刷新一下文档数据库，就可以使用该字体配置文件了
>
> [67] 这个配置文件使用 Adobe Creative Suite 4 中附带的四种字体
>
> [68] 前面给出的 `ctx` 文件已设置此变量

Windows 下安装

首先下载 context-setup-mswin 和 ruby

安装 ruby ；解压 context-setup-mswin ，运行 `first-setup.bat` [69]

假设 conTeXt 安装在 `D:\context` 目录下，设置环境变量[70]，在 `PATH` 项中添加路径：

```
;D:\context\tex\texmf-mswin\bin
```

将以下内容保存为 `environment.reg` ，双击导入注册表

```
Windows Registry Editor Version 5.00

[HKEY_CURRENT_USER\Environment]
"CYGWIN"="nontsec"
"OSFONTDIR"="C:\\WINDOWS\\Fonts"
"TEXMF"="{$TEXPATH\\texmf,$TEXPATH\\texmf-local,$TEXPATH\\texmf-context,$TEXPATH\\texmf-m
"TEXMFCACHE"="$TEXPATH\\texmf-cache"
"TEXMFCNF"="$TEXPATH\\texmf{-local,-context,}/web2c"
"TEXMFOS"="$TEXPATH\\texmf-mswin"
"TEXPATH"="D:\\context\\tex"
```

将打印字体配置文件复制到 `D:\context\tex\texmf-local\tex\context\third` 目录下[71]

使用命令 `context --generate` 刷新文档数据库[72]

最后，使用命令 `context a.tex` 或者 `texexec a.tex` 生成 PDF 文档

> [69] 这是一个自动下载、升级的脚本
>
> [70] 我的电脑:右键→属性→高级→环境变量
>
> [71] 默认的配置文件需要4个 Adobe 的字体，如果没有这四个字体，可以将配置文件中使用到的 Adobe 字体改为系统中的字体
>
> [72] 需要在 Windows 命令行中执行

项目管理

为了养成良好的习惯，我们把 ConTeXt 文档拆分到几个单独的 `.tex` 文件中，这样维护起来比较方便。

首先是主文档 `product.tex` [73]，生成 PDF 只要编译此文件即可

```
%%%%%此文件使用 product 环境，起始声明
\startproduct{}

%%%导言区使用 \environment 载入文件
%载入样式文件 style.tex
\environment style

%载入词汇定义文件 gloss.tex
\environment gloss

%%%正文起始
\starttext

%%%正文区使用 \component 载入文件
%封面 cover.tex
\component cover

%目录
\title{目录}
\placecontent

%%正文内容

%载入章节 1.tex 2.tex 3.tex
\component 1
\component 2
\component 3

%%%正文结束
\stoptext

%%%%% product 环境结束声明
\stopproduct
```

在导言区载入的文件，要使用 `environment` 环境，例如样式定义文件 `style.tex`

```
\startenvironment{}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%               中文设置                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\usetypescriptfile[zhfonts]     %加载打印字体配置文件(typescript) zhfonts.tex
\usetypescript[myscript]        %使用打印字体配置文件中定义打印字体的脚本 myscript
\setupbodyfont[myfont,rm,11pt]  %设置正文字体

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%              正文   标题               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 启用颜色模式， 设置链接文本颜色
\setupcolors[state=start]
\definecolor[linktext][darkred]
\setupinteraction[state=start,color=linktext]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%正文
\setupindenting[always,2em,first]                  %设置中文缩进格式(首行缩进两字)
\setupheads[indentnext=yes]                        %每节的首段也要缩进

\setupinterlinespace[big]                          %设置行距(big=1.5倍)
\setupwhitespace[small]                           %设置段间距[small, medium, big]

%标题
\setupheads[indentnext=yes]
\setuphead
[chapter]
[style=\bfc,header=empty,footer=empty]
\setuphead
```

```
[section]
[style=\bfa]
\setuphead
[title]
[style=\bfb,header=empty,foote=empty]
\setuphead
[subsubject]
[style=\bf]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%               页面设置               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%定义页面尺寸为 SCREEN
\definepapersize[mySCREEN][width=21cm,height=29.7cm]
%纸张尺寸，通常和页面尺寸相同。(除非在印刷用纸上实现多页排版)
\setuppapersize[SCREEN][mySCREEN]

%布局
\setuplayout
[width=fit,
height=middle,
leftmargin=3cm,
rightmargin=3cm,
backspace=4cm,
topspace=.5cm,
headerdistance=.4cm,
footerdistance=.4cm,
header=1cm,
footer=1cm]

%去掉页眉正中 自动添加的页码
\setuppagenumbering
[style=\tfx,location=]

%页眉
\def\CurrentChapter{%
第 \headnumber[chapter]\ 章%
\hbox to 2em{}%
\getmarking[chapter]%
}
\def\CurrentSection{%
\headnumber[section]%
\hbox to 2em{}%
\getmarking[section]%
}
\setupheadertexts
[\CurrentChapter][pagenumber]
[pagenumber][\CurrentSection]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%边注
\setupinmargin[left,right][style=\tfx]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%PDF 阅读器中自适应页宽
\setupinteraction[state=start,openaction=FitWidth]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%              目录  书签              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 启用书签功能
\setupinteraction[state=start]
\setupinteractionscreen[option=bookmark]
\placebookmarks[chapter,section,subsection][chapter]

%%%经典目录样式
%% turn off numbering of some levels
%\setuphead[subsection][number=no]
%\setuphead[subsubsection][number=no]
```

```
%% TOC
%% level=4, \subsubsubsections are not listed in TOC
%% alternative=c, space to the page number is filled with dots
%\setupcombinedlist[content][level=4,alternative=c]
%%\setuplist[chapter][width=5mm,style=bold]
%\setuplist[section][width=10mm,style=bold]
%\setuplist[subsection][width=20mm]
%% pagestyle=normal for changing the appearance of pagenumber
%\setuplist[subsubsection][width=20mm,style=slanted,pagestyle=normal]

%目录样式
\def\ChapterNumber#1{\doiftext{#1}{第\;#1\;章\quad}}
\setuplist
[chapter]
[alternative=a,
before={\page[preference]\blank},
after=\blank,
style=bold,
width=fit,
pagestyle=boldslanted,
pagenumber=no,
numbercommand=\ChapterNumber]

\def\PageNumber#1{\color[darkgray]{#1}.}
\setuplist
[section]
[alternative=d,
style=small,
pagecommand=\PageNumber,
pagestyle=\itx]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\stopenvironment
```

词汇定义文件 `gloss.tex`

```
\startenvironment{}

%使用命令 \hello{mom}
%得到 Good morning mom
\define[1]\hello{Good morning, #1}

%使用命令 \lxsc
%得到 《开源世界旅行手册》
\define\lxsc{《开源世界旅行手册》}

\stopenvironment
```

章节放在单独的文件中，例如 `1.tex`

```
\startcomponent{}

\chapter{第一章}

ConTeXt 组件文档

\stopcomponent
```

> [73] 文件名任意

语法简介

空白距离

TEX 将空格和制表符等空白字符视为相同的空白距离（space），多个连续的 空白字符 等同为一个空白字符, 每行开始的空白字符将被忽略， 而单个的 换行符 被视为一空格。

TEX 使用空行(两个或以上换行符)来分隔段落。如同空格一样，多个空行所起的作用和一个空行是相同的。

以下为示例代码：

```
TEX 将空格和制表符等空白字符视为相同的空白距离（space），
多个连续的  空白字符          等同为一个空白字符,
        每行开始的空白字符将被忽略，
而单个的  换行符
被视为一空格。

TEX 使用空行(两个或以上换行符)来分隔段落。如同空格一样，多个空行所起的作用和一个空行是相同的。

以下为示例代码：
```

可以使用 `\blank` [74]命令插入空行，使用 `\\` 插入空格

特殊字符

以下为 TeX 的保留字符：

保留字符	禁止解析	说明
\	\type{}	命令引导符
%	\%	注释
$	$	数学公式
^	\^	上标
_	_	下标
#	#	
&	\&	
~	~	
{	{	命令参数
}	}	命令参数

`\backslash` 可以得到数学符号中的 ****，而 `\\` 通常在标题、页眉/页脚、边注等环境中作为换行符

命令

TEX 命令以反斜线“`\`”引导，以任意非字母字符结束

命令可以使用花括号或方括号作为参数，例如：

```
\command[选项,设置,……]{操作对象}
```

命令分为两种：普通命令和环境命令。普通命令可以用于行内；环境命令包含起始声明和结束声明，用于多行。普通命令和环境命令可以互相嵌套

```
\begin{itemize}                                    %环境命令起始声明
\item 列表项                          %普通命令
\item 列表项
\end{itemize}                                         %环境命令结束声明
```

> [74] `\blank[medium*4]` 插入4个中等高度空行

标题

conTeXt 中，章节以下列命令开始，并使用 **{}** 指定标题，例如 `\chapter{第一章}`：

章节	编号	不编号
部	\part	
章	\chapter	\title
节	\section	\subject
小节	\subsection	\subsubject
	\subsubsection	\subsubsubject

- 章节无需结束声明，遇到下一个章节命令，本章节结束

使用方括号给章节定义标记

```
\title[引用]{标题} %定义标记
\at{page}[引用]   %使用 `\at` 命令引用
```

列表

使用以下命令插入列表：

```
\startitemize[R,packed,broad]  %列表起始声明 [选项]
\item 列表项一                  %列表项使用 **\item** 命令
\item 列表项二
\item 列表项三
\stopitemize                    %列表结束声明
```

第一个选项为列表项分隔符(前缀)：

选项	样式
1	圆点
2	短划线
3	星号
4	三角形
5	梅花
6	空心圆
n	数字
a	小写字母
A	大写字母
r	小写罗马数字
R	大写罗马数字

后两个为格式选项

选项	格式
standard	标准
packed	压缩列表项垂直间距
serried	压缩分隔符和文本的间距
joinedup	压缩列表与正文间距
broad	扩展分隔符和文本的间距
inmargin	将分隔符放在而边缘
atmargin	
stopper	
columns	
intro	
continue	

columns 选项需要参数

```
\startitemize[columns,three,broad]
```

数学公式

数学环境

```
\placeformula[1]                %给数学公式编号[引用标记]
\startformula            %数学环境起始声明
y=x^2
\stopformula            %数学环境结束声明

这是行内的数学环境 $\int_0^1 x^2 dx$
```

运算符

可以直接使用的基本运算符有：

```
+ - = < >
```

数学符号	命令
±	\pm
×	\times
÷	\div
*	\ast
★	\star
●	\bullet
○	\circ
·	\cdot
≤	\leq
《	\ll
≥	\geq
》	\gg
≡	\equiv
～	\sim
≈	\approx
≠	\neq
∩	\cap
∪	\cup
∈	\in

上标、下标

使用 `^` 输入上标

```
勾股定理 $3^2 + 4^2 = 5^2$    %`^2`为指数
二的十次方 $2^{10} = 1024$    %如果指数不止一位，要使用`{}`括起来
```

打印字体配置文件

conTeXt 使用以下方式定义字体：

```
\definefont                                   %使用此命令定义字体。这三句可以写在一行
[Song]                                        %第一个参数为字体别名
[{`name`:Adobe Song Std*zh} at 14pt]  %字体定义：使用14号的 Adobe 中文宋体
%`name`使用字体名称；`file`使用字体文件

\starttext
\Song 我使用的是 Adobe 宋体               %使用字体别名指定该行体样式
\stoptext
```

使用这种方式指定字体比较繁琐，推荐建立一个打印字体配置文件(`typescript`)，如 `zhfonts.tex` (如何使用 `zhfonts.tex` 参阅“字体设置”一节)

`.tex` 源文件中使用以下命令定义字体：

```
\usetypescriptfile[zhfonts]     %加载打印字体配置文件 `zhfonts.tex`
\usetypescript[myscript]        %使用打印字体配置文件中定义打印字体的脚本 `myscript`
\setupbodyfont[myfont,ss,11pt]  %使用脚本预定义的 `myfont` 字体： 字型为 `ss` ，大小为 11pt
```

通常有三种字型的字体：衬线、非衬线、等宽，衬线字型笔划起始和结束的地方有一些修饰，非衬线字体笔划为无修饰的线条，等宽字体所有的字符宽度相同

而一种字体，无论属于何种字型，都可能有几种样式，例如：普通、粗体、斜体、斜粗体

我们在打印字体配置文件中，首先定义字型，然后为每种字型定义样式。

在 `zhfonts.tex` 文件中定义打印字体 `myfont` 的字型：

```
\starttypescript[myscript]                    %定义打印字体的脚本为 `myscritp`
\definetypeface[myfont][ss][sans][zhfont]     %将 `myfont` 的字型 `ss` 定义为 `[sans][zhf
\stoptypescript                                         %结束定义
```

`[sans][zhfont]` 为非衬线字型，给它指定各种样式，在上面定义之前写入以下配置：

```
\starttypescript [sans][zhfont]    %定义字型 `[sans][zhfont]`
\definefontsynonym    %此命令定义字体别名
      %定义该字型的普通样式 `[Sans]` 为 `[ZhSans]`，备用字体为 [fallbacks=WesternSans]
      [Sans][ZhSans][fallbacks=WesternSans]
\definefontsynonym
      %定义该字型的粗体样式 `[SansBold]` 为 `[ZhSansBold]`，备用字体为 [fallbacks=WesternSansBo
      [SansBold][ZhSansBold][fallbacks=WesternSansBold]
\definefontsynonym
        %定义斜体
    [SansItalic][ZhSansItalic][fallbacks=WesternSansItalic]
\definefontsynonym
        %定义斜粗体
    [SansBoldItalic][ZhSansBoldItalic][fallbacks=WesternSansBoldItalic]
\stoptypescript
```

上一步定义使用别名定义别名，写起来比较麻烦，但是却可以选择备用字体。接下来将各种样式定义为实际的字体：

```
\starttypescript [sans][zhfont]
%定义四种样式实际使用的字体
%`name`使用字体名称；`file`使用字体文件
%`[features=zh]` 字体使用 `zh` 属性，还未定义
\definefontsynonym
    [ZhSans][name:AdobeKaitiStd-Regular][features=zh]
\definefontsynonym
    [ZhSansBold][name:AdobeHeitiStd-Regular][features=zh]
\definefontsynonym
    [ZhSansItalic][name:AdobeKaitiStd-Regular][features=zh]
\definefontsynonym
    [ZhSansBoldItalic][name:AdobeHeitiStd-Regular][features=zh]
%定义备用字体的四种样式
\definefontfallback
    [WesternSans][name:TeXGyreAdventor-Regular][0x0000-0x0400][force=yes]
\definefontfallback
    [WesternSansBold][name:TeXGyreAdventor-Bold][0x0000-0x0400][force=yes]
\definefontfallback
    [WesternSansItalic][name:TeXGyreAdventor-Italic][0x0000-0x0400][force=yes]
\definefontfallback
    [WesternSansBoldItalic][name:TeXGyreAdventor-BoldItalic][0x0000-0x0400][force=yes]

\definefontsynonym
 [Sans][ZhSans][fallbacks=WesternSans]
\definefontsynonym
 [SansBold][ZhSansBold][fallbacks=WesternSansBold]
\definefontsynonym
 [SansItalic][ZhSansItalic][fallbacks=WesternSansItalic]
\definefontsynonym
 [SansBoldItalic][ZhSansBoldItalic][fallbacks=WesternSansBoldItalic]
\stoptypescript
```

最后，定义字体属性 `zh`

```
\definefontfeature[zh][mode=node,script=hang,lang=zhs]
```

完整的 `zhfonts.tex` 文件，参阅“字体设置”一节

部分 III. 景观

目录

44. FVWM

第 30 章 终极 Shell -- ZSH

目录

Zsh 简介

zsh: The last shell you'll ever need!

Z 是最后一个字母，所以它是终极 Shell。

zsh 拥有很多非常实用的功能

提示符

Zsh 的命令提示符令人印象深刻。它支持右侧对齐的提示符，并且可以配置成这个样子的：

自动补全

Zsh 的自动补全功能十分强大，可以根据上下文补全命令、选项、文件名、进程、用户名、变量、权限符等。不需要记忆繁多的快捷键，只要按 `Tab` 就可以了。

路径别名

Zsh 拥有贴心的路径别名。假设有一个很长的路径，例如 `/home/lighttpd/html` ，可以把这个路径命名为 `~www` 。

```
alias ls='ls -F --color=auto'
alias ll='ls -l'
alias grep='grep --color=auto'
alias ee='emacsclient -t'
##################################################################
#Alias Directories
#hash -d dir_name="dir_path"
#cd ~dir_name
hash -d WWW="/home/lighttpd/html"
hash -d ARCH="/mnt/arch"
hash -d PKG="/var/cache/pacman/pkg"
hash -d E="/etc/env.d"
hash -d C="/etc/conf.d"
hash -d I="/etc/rc.d"
hash -d X="/etc/X11"
hash -d BK="/home/r00t/config_bak"


##for Emacs
                                                    106,1          93%
```

```
ic
r00t@myhost 08-09-13 14:24  /home/r00t
>>cd ~WWW/
 -- user --
 -- named directory --
ARCH    C       ftp     mail    r00t
avahi   daemon  hal     mysql   root
bin     dbus    http    nobody  WWW
BK      E       I       PKG     X
```

```
r00t@myhost 08-09-13 14:34  /home/lighttpd/html
>>pwd
/home/lighttpd/html
r00t@myhost 08-09-13 14:34  /home/lighttpd/html
>>
```

也可以使用 **cd** 历史纪录，`cd -(TAB)`

```
r00t@myhost 08-10-29 3:10  /etc/rc.d
>>cd -0
 -- directory stack --
0 -- /home/r00t
1 -- /home/lighttpd/html
2 -- /var/cache/pacman/pkg
```

后缀别名

zsh 还可以设定后缀别名

```
#alias 默认为命令别名
alias ls='ls -F --color=auto'
#使用选项 -s 定义后缀别名，键入扩展名为“xml”的文件名，自动使用 emacs 打开
alias `-s` xml=emacs
```

键绑定

Zsh 默认使用 Emacs风格 的键绑定，习惯 Bash 键绑定的朋友无需重新适应。Zsh 兼容大多数主流 Shell，像 Bash、Csh 等。

错误校正

Zsh 还拥有错误校正的功能

```
┌(r00t@myhost:pts/0)                                                  (~BK)┐
└(21:40:%)── cd /etc/x11                                       ──(二, 9月16)┘
 -- directory --
etc/
```

-- directory -- 是补全类型提示

/etc/x11 [tab] 后被修正为 /etc/X11 ，补全类型提示变成了 -- corrections --

```
┌(r00t@myhost:pts/0)                                                  (~BK)┐
└(21:40:%)── cd /etc/X11/                                      ──(二, 9月16)┘
 -- corrections --
X11/
```

这个功能不是单纯的修正大小写，而是各种拼写错误。 比如说上面的例子，如果输入的是 11 或者 s11，它一样会修正为 X11

有一个前提，就是每次修正，只允许有一处字符错误。 两个以上的错误，除非可以匹配其它的选项，否则就不能修正， 12 就不能修正为 X11 ，除非候选里有 X12、Y12、Z12……

在配置文件里找到这一行，修改容错字数

```
zstyle ':completion:*:approximate:*' max-errors 1 numeric
```

Zsh 的强大特性

重定向功能

示例：

重定向 stdout 和 stderr 到 file	command |& >file
同时重定向到多个文件	command >file.1 >file.2

补全类型控制

mpg123 [tab] , 候选菜单中只出现扩展名为 .mp3 .MP3 的文件：

```
zstyle ':completion:*:*:mpg123:*' file-patterns \
  '*.(mp3|MP3):mp3\ files *(-/):directories'

zstyle ':completion:*:*:ogg123:*' file-patterns \
  '*.(ogg|OGG):ogg\ files *(-/):directories'
```

计算器

zsh 可以当作计算器使用

```
#载入数学函数模块 可以进行一些比较高级的运算
#（也可以将此句写在配置文件中）
$ zmodload zsh/mathfunc
#`$((`数学表达式`))` 进行运算，使用 echo 显示结果
$ echo $(( sin(1/4.0)**2 + cos(1/4.0)**2 - 1 ))
  -1.1102230246251565e-16
$ echo $(( pi = 4.0 * atan(1.0) ))
  3.1415926535897931
$ echo $(( f = sin(0.3) ))
  0.29552020666133955
$ print $((1e12 * rand48()))
  847909677310.23413
$ print $(( rand48(seed) ))
  0.01043488334700271
```

命令替换

```
# bash 中使用这种形式
$ emacs find . -name "*.html"
# zsh 同样支持，并可以使用以下形式
$ emacs `$(`ls **/*.html`)`

# zsh 还可以将命令结果生成临时文件，并返回文件名
#支持更复杂的输出和过滤。例如，比较 `new/` 和 `old/` 两个文件夹的内容
$ diff `=(`ls new/`)`  `=(`ls old/`)`

#查看生成的临时文件
ls =()
```

Zsh 配置文件

例 **30.1. Zsh** 配置文件 `.zshrc`

```
#color{{{
autoload colors
colors

for color in RED GREEN YELLOW BLUE MAGENTA CYAN WHITE; do
eval _$color='%{$terminfo[bold]$fg[${(L)color}]%}'
eval $color='%{$fg[${(L)color}]%}'
```

```
(( count = $count + 1 ))
done
FINISH="%{$terminfo[sgr0]%}"
#}}}

#命令提示符 {{{
RPROMPT=$(echo "$RED%D %T$FINISH")
PROMPT=$(echo "$BLUE%M$GREEN%/
$CYAN%n $_YELLOW>>>$FINISH ")
#}}}

#标题栏、任务栏样式{{{
case $TERM in (*xterm*|*rxvt*|(dt|k|E)term)
   precmd () { print -Pn "\e]0;%n@%M//%/\a" }
   preexec () { print -Pn "\e]0;%n@%M//%/\ $1\a" }
   ;;
esac
#}}}

#关于历史纪录的配置 {{{
#历史纪录条目数量
export HISTSIZE=10000
#注销后保存的历史纪录条目数量
export SAVEHIST=10000
#历史纪录文件
#export HISTFILE=~/.zhistory
#以附加的方式写入历史纪录
setopt INC_APPEND_HISTORY
#如果连续输入的命令相同，历史纪录中只保留一个
setopt HIST_IGNORE_DUPS
#为历史纪录中的命令添加时间戳
setopt EXTENDED_HISTORY

#启用 cd 命令的历史纪录，cd -[TAB]进入历史路径
setopt AUTO_PUSHD
#相同的历史路径只保留一个
setopt PUSHD_IGNORE_DUPS

#在命令前添加空格，不将此命令添加到纪录文件中
#setopt HIST_IGNORE_SPACE
#}}}

#每个目录使用独立的历史纪录{{{
cd() {
    builtin cd "$@"                             # do actual cd
    fc -W                                       # write current history  file
    local HISTDIR="$HOME/.zsh_history$PWD"      # use nested folders for history
        if  [ ! -d "$HISTDIR" ] ; then          # create folder if needed
            mkdir -p "$HISTDIR"
        fi
        export HISTFILE="$HISTDIR/zhistory"     # set new history file
    touch $HISTFILE
    local ohistsize=$HISTSIZE
        HISTSIZE=0                              # Discard previous dir's history
        HISTSIZE=$ohistsize                     # Prepare for new dir's history
    fc -R                                       #read from current histfile
}
mkdir -p $HOME/.zsh_history$PWD
export HISTFILE="$HOME/.zsh_history$PWD/zhistory"

function allhistory { cat $(find $HOME/.zsh_history -name zhistory) }
function convhistory {
            sort $1 | uniq |
            sed 's/^:\([ 0-9]*\):[0-9]*;\(.*\)/\1::::::\2/' |
            awk -F"::::::" '{ $1=strftime("%Y-%m-%d %T",$1) "|"; print }'
}
#使用 histall 命令查看全部历史纪录
function histall { convhistory =(allhistory) |
            sed '/^.\{20\} *cd/i\\' }
#使用 hist 查看当前目录历史纪录
function hist { convhistory $HISTFILE }
```

```
#全部历史纪录 top44
function top44 { allhistory | awk -F':[ 0-9]*:[0-9]*;' '{ $1="" ; print }' | sed 's/ /\n/

#}}}

#杂项 {{{
#允许在交互模式中使用注释  例如：
#cmd #这是注释
setopt INTERACTIVE_COMMENTS

#启用自动 cd，输入目录名回车进入目录
#稍微有点混乱，不如 cd 补全实用
#setopt AUTO_CD

#扩展路径
#/v/c/p/p => /var/cache/pacman/pkg
setopt complete_in_word

#禁用 core dumps
limit coredumpsize 0

#Emacs风格 键绑定
bindkey -e
#设置 [DEL]键 为向后删除
bindkey "\e[3~" delete-char

#以下字符视为单词的一部分
WORDCHARS='*?_-[]~=&;!#$%^(){}<>'
#}}}

#自动补全功能 {{{
setopt AUTO_LIST
setopt AUTO_MENU
#开启此选项，补全时会直接选中菜单项
#setopt MENU_COMPLETE

autoload -U compinit
compinit

#自动补全缓存
#zstyle ':completion::complete:*' use-cache on
#zstyle ':completion::complete:*' cache-path .zcache
#zstyle ':completion:*:cd:*' ignore-parents parent pwd

#自动补全选项
zstyle ':completion:*' verbose yes
zstyle ':completion:*' menu select
zstyle ':completion:*:*:default' force-list always
zstyle ':completion:*' select-prompt '%SSelect:  lines: %L  matches: %M  [%p]'

zstyle ':completion:*:match:*' original only
zstyle ':completion::prefix-1:*' completer _complete
zstyle ':completion:predict:*' completer _complete
zstyle ':completion:incremental:*' completer _complete _correct
zstyle ':completion:*' completer _complete _prefix _correct _prefix _match _approximate

#路径补全
zstyle ':completion:*' expand 'yes'
zstyle ':completion:*' squeeze-shlashes 'yes'
zstyle ':completion::complete:*' '\\'

#彩色补全菜单
eval $(dircolors -b)
export ZLSCOLORS="${LS_COLORS}"
zmodload zsh/complist
zstyle ':completion:*' list-colors ${(s.:.)LS_COLORS}
zstyle ':completion:*:*:kill:*:processes' list-colors '=(#b) #([0-9]#)*=0=01;31'

#修正大小写
zstyle ':completion:*' matcher-list '' 'm:{a-zA-Z}={A-Za-z}'
#错误校正
zstyle ':completion:*' completer _complete _match _approximate
```

```
zstyle ':completion:*:match:*' original only
zstyle ':completion:*:approximate:*' max-errors 1 numeric

#kill 命令补全
compdef pkill=kill
compdef pkill=killall
zstyle ':completion:*:*:kill:*' menu yes select
zstyle ':completion:*:*:*:*:processes' force-list always
zstyle ':completion:*:processes' command 'ps -au$USER'

#补全类型提示分组
zstyle ':completion:*:matches' group 'yes'
zstyle ':completion:*' group-name ''
zstyle ':completion:*:options' description 'yes'
zstyle ':completion:*:options' auto-description '%d'
zstyle ':completion:*:descriptions' format $'\e[01;33m -- %d --\e[0m'
zstyle ':completion:*:messages' format $'\e[01;35m -- %d --\e[0m'
zstyle ':completion:*:warnings' format $'\e[01;31m -- No Matches Found --\e[0m'
zstyle ':completion:*:corrections' format $'\e[01;32m -- %d (errors: %e) --\e[0m'

# cd ~ 补全顺序
zstyle ':completion:*:-tilde-:*' group-order 'named-directories' 'path-directories' 'user
#}}}

##行编辑高亮模式 {{{
# Ctrl+@ 设置标记，标记和光标点之间为 region
zle_highlight=(region:bg=magenta #选中区域
               special:bold      #特殊字符
               isearch:underline)#搜索时使用的关键字
#}}}

##空行(光标在行首)补全 "cd " {{{
user-complete(){
    case $BUFFER in
        "" )                       # 空行填入 "cd "
            BUFFER="cd "
            zle end-of-line
            zle expand-or-complete
            ;;
        "cd --" )                  # "cd --" 替换为 "cd +"
            BUFFER="cd +"
            zle end-of-line
            zle expand-or-complete
            ;;
        "cd +-" )                  # "cd +-" 替换为 "cd -"
            BUFFER="cd -"
            zle end-of-line
            zle expand-or-complete
            ;;
        * )
            zle expand-or-complete
            ;;
    esac
}
zle -N user-complete
bindkey "\t" user-complete
#}}}

##在命令前插入 sudo {{{
#定义功能
sudo-command-line() {
    [[ -z $BUFFER ]] && zle up-history
    [[ $BUFFER != sudo\ * ]] && BUFFER="sudo $BUFFER"
    zle end-of-line                 #光标移动到行末
}
zle -N sudo-command-line
#定义快捷键为： [Esc] [Esc]
bindkey "\e\e" sudo-command-line
#}}}

#命令别名 {{{
alias -g cp='cp -i'
```

```
alias -g mv='mv -i'
alias -g rm='rm -i'
alias -g ls='ls -F --color=auto'
alias -g ll='ls -l'
alias -g grep='grep --color=auto'
alias -g ee='emacsclient -t'

#[Esc][h] man 当前命令时，显示简短说明
alias run-help >&/dev/null && unalias run-help
autoload run-help

#历史命令 top10
alias top10='print -l  ${(o)history%% *} | uniq -c | sort -nr | head -n 10'
#}}}

#路径别名 {{{
#进入相应的路径时只要 cd ~xxx
hash -d WWW="/home/lighttpd/html"
hash -d ARCH="/mnt/arch"
hash -d PKG="/var/cache/pacman/pkg"
hash -d E="/etc/env.d"
hash -d C="/etc/conf.d"
hash -d I="/etc/rc.d"
hash -d X="/etc/X11"
hash -d BK="/home/r00t/config_bak"
#}}}

##for Emacs {{{
#在 Emacs终端 中使用 Zsh 的一些设置 不推荐在 Emacs 中使用它
if [[ "$TERM" == "dumb" ]]; then
setopt No_zle
PROMPT='%n@%M %/
>>'
alias ls='ls -F'
fi
#}}}

#{{{自定义补全
#补全 ping
zstyle ':completion:*:ping:*' hosts 192.168.128.1{38,} www.g.cn \
       192.168.{1,0}.1{{7..9},}

#补全 ssh scp sftp 等
my_accounts=(
{r00t,root}@{192.168.1.1,192.168.0.1}
kardinal@linuxtoy.org
123@211.148.131.7
)
zstyle ':completion:*:my-accounts' users-hosts $my_accounts
#}}}

#{{{ F1 计算器
arith-eval-echo() {
  LBUFFER="${LBUFFER}echo \$(( "
  RBUFFER=" ))$RBUFFER"
}
zle -N arith-eval-echo
bindkey "^[[11~" arith-eval-echo
#}}}

####{{{
function timeconv { date -d @$1 +"%Y-%m-%d %T" }

# }}}

## END OF FILE #################################################################
# vim:filetype=zsh foldmethod=marker autoindent expandtab shiftwidth=4
```

这是一种效果超炫的提示符，把上面与提示符相关的配置语句注释掉，加入下面代码

```
#线框型提示符

function precmd {

local TERMWIDTH
(( TERMWIDTH = ${COLUMNS} - 1 ))

###
# 如果路径太长，截短它

FILLBAR=""
PWDLEN=""

#统计 %~ 双字节字符
local count_db_wth_char="$( echo ${^${(%):-%~}} | sed 's/\(.\)/\1\n/g' | grep -c \[\^\!-\
local promptsize=${#${(%):---(%n@%m:%l)---()--}}
local pwdsize=${#${(%):-%~}}+$count_db_wth_char

if [[ "$promptsize + $pwdsize" -gt $TERMWIDTH ]]; then
((PWDLEN=$TERMWIDTH - $promptsize))
else
FILLBAR="\${(l.(($TERMWIDTH - ($promptsize + $pwdsize)))..${HBAR}.)}"
fi

###
# Get APM info.

#if which ibam > /dev/null; then
#APM_RESULT=`ibam --percentbattery`
#elif which apm > /dev/null; then
#APM_RESULT=`apm`
#fi
}

setopt extended_glob
preexec () {
if [[ "$TERM" == "screen" ]]; then
local CMD=${1[(wr)^(*=*|sudo|-*)]}
echo -n "\ek$CMD\e\\"
fi
}

setprompt () {
###
# Need this so the prompt will work.

setopt prompt_subst

###
# See if we can use colors.

autoload colors zsh/terminfo
if [[ "$terminfo[colors]" -ge 8 ]]; then
colors
fi
for color in RED GREEN YELLOW BLUE MAGENTA CYAN WHITE; do
eval $color='%{$terminfo[bold]$fg[${(L)color}]%}'
eval LIGHT_$color='%{$fg[${(L)color}]%}'
(( count = $count + 1 ))
done
NO_COLOUR="%{$terminfo[sgr0]%}"

###
# See if we can use extended characters to look nicer.

typeset -A altchar
set -A altchar ${(s..)terminfo[acsc]}
SET_CHARSET="%{$terminfo[enacs]%}"
SHIFT_IN="%{$terminfo[smacs]%}"
SHIFT_OUT="%{$terminfo[rmacs]%}"
HBAR=${altchar[q]:--}
```

```
#HBAR=" "
ULCORNER=${altchar[l]:--}
LLCORNER=${altchar[m]:--}
LRCORNER=${altchar[j]:--}
URCORNER=${altchar[k]:--}

###
# Decide if we need to set titlebar text.

case $TERM in
xterm*)
TITLEBAR=$'%{\e]0;%(!.-=*[ROOT]*=- | .)%n@%m:%~ | ${COLUMNS}x${LINES} | %y\a%}'
;;
screen)
TITLEBAR=$'%{\e_screen \005 (\005t) | %(!.-=[ROOT]=- | .)%n@%m:%~ | ${COLUMNS}x${LINES} |
;;
*)
TITLEBAR=''
;;
esac

###
# Decide whether to set a screen title
if [[ "$TERM" == "screen" ]]; then
STITLE=$'%{\ekzsh\e\\%}'
else
STITLE=''
fi

###
# APM detection

#if which ibam > /dev/null; then
#APM='$RED${${APM_RESULT[(f)1]}[(w)-2]}%%(${${APM_RESULT[(f)3]}[(w)-1]})$LIGHT_BLUE:'
#elif which apm > /dev/null; then
#APM='$RED${APM_RESULT[(w)5,(w)6]/\% /%%}$LIGHT_BLUE:'
#else
APM=''
#fi

###
# Finally, the prompt.

PROMPT='$SET_CHARSET$STITLE${(e)TITLEBAR}\
$CYAN$SHIFT_IN$ULCORNER$BLUE$HBAR$SHIFT_OUT(\
$GREEN%(!.%SROOT%s.%n)$GREEN@%m:%l\
$BLUE)$SHIFT_IN$HBAR$CYAN$HBAR${(e)FILLBAR}$BLUE$HBAR$SHIFT_OUT(\
$MAGENTA%$PWDLEN<...<%~%<<\
$BLUE)$SHIFT_IN$HBAR$CYAN$URCORNER$SHIFT_OUT\

$CYAN$SHIFT_IN$LLCORNER$BLUE$HBAR$SHIFT_OUT(\
%(?..$LIGHT_RED%?$BLUE:)\
${(e)APM}$YELLOW%D{%H:%M}\
$LIGHT_BLUE:%(!.$RED.$WHITE)%#$BLUE)$SHIFT_IN$HBAR$SHIFT_OUT\
$CYAN$SHIFT_IN$HBAR$SHIFT_OUT\
$NO_COLOUR '

RPROMPT=' $CYAN$SHIFT_IN$HBAR$BLUE$HBAR$SHIFT_OUT\
($YELLOW%D{%a,%b%d}$BLUE)$SHIFT_IN$HBAR$CYAN$LRCORNER$SHIFT_OUT$NO_COLOUR'

PS2='$CYAN$SHIFT_IN$HBAR$SHIFT_OUT\
$BLUE$SHIFT_IN$HBAR$SHIFT_OUT(\
$LIGHT_GREEN%_$BLUE)$SHIFT_IN$HBAR$SHIFT_OUT\
$CYAN$SHIFT_IN$HBAR$SHIFT_OUT$NO_COLOUR '
}

setprompt
```

第 31 章 完美工作站 Archlinux

目录

简介

惜Fedora、SuSE，有点臃肿

gentoo、LFS，略显麻烦

一代天骄，Ubuntu

只知免费送光盘

深孚众望的 Ubuntu 变得越来越臃肿，越来越“Windows”；Gentoo 虽然能够灵活定制，不过所有的软件都使用源代码编译安装，却也有些过犹不及

Archlinux，既能够像 Ubuntu 那样使用二进制包(pacman)便捷的安装，又能够像 Gentoo 那样灵活定制(ABS)，是一个十分均衡的发行版，它还有许多夺目的亮点：

简约

Archlinux 信奉 UNIX 传统的 KISS 哲学，安装和配置十分容易，文件系统结构布局清晰

轻快

Archlinux 为 i686 进行优化，无论是系统的启动，还是运行程序，都给人以快马轻裘的感觉

灵活

Archlinux 安装后只有一个最基础的系统，你可以在这个基础上如搭积木般，使用 pacman 安装软件

对于想定制系统的朋友，在最小化系统上作加法，要比在臃肿系统中作减法来得容易

前卫

Archlinux 总是尽量保持系统中的软件为最新版本

编译系统

Archlinux 使用 ABS 系统简化琐屑的编译过程，并且用户可以通过 AUR 分享、交流 PKGBUILD

> 提示：`KISS` : Keep It Simple, Stupid!

Archlinux 是完美的 工作站/桌面，但并不推荐使用它作服务器，因为太过前卫有可能会导致系统的不稳定

安装基本系统

大致上，安装 Linux 分为几个阶段：

前期准备

通过各种方式获取安装介质，进入安装作业平台。比如下载 ISO 映像刻录光盘，使用光盘引导，启动安装程序

选择安装源

选择通过安装介质安装还是通过网络下载的方式进行安装。如果通过网络安装，可能需要配置网络

准备磁盘

操作系统最终是安装在磁盘上的，所以要分割磁盘、格式化分区、挂载

选择、安装软件包

选择需要的软件。有些发行版如 Ubuntu LiveCD 安装时不能选择软件

配置系统

对系统进行各种配置，使其能良好运行

安装引导程序

以便启动时可以引导系统

新建用户

使用 root 用户进行操作存在一定风险，尽量建立一个用于日常操作的用户

前期准备

首先下载 Archlinux 安装介质。

有两种安装介质： ISO 和 USB ，其中 ISO 为光盘映像，用来刻录安装光盘； USB 为磁盘映像，可以恢复到 USB 移动存储设备

每种安装介质包含两种目标架构： i686 和 x86_64 ，其中 i686 适用于 奔腾II™ 以上级别 CPU ； x86_64 只适用于 64 位 CPU

每种目标架构又包含两个版本： CORE 和 FTP Install ，其中 FTP Install 只包含作业平台，需要通过网络下载软件包；CORE 除了作业平台，还包含基本系统[75]

假设您下载了 archlinux-2008.06-core-i686.iso ，刻录并使用它引导，您会看到以下界面

```
Copyright 2002 - 2007 Judd Vinet <jvinet@zeroflux.org>
Distributed under the GNU General Public License (GPL)

ISOLINUX BOOT
Creation Tool: 'mkbootcd' written by Tobias Powalowski <tpowa@archlinux.org>

INSTALLATION SYSTEM
Arch Linux Don't Panic
Kernel: 2.6.22-ARCH
Architecture: i686
Creation Date: Wed Oct  3 13:40:13 UTC 2007

Available boot options (no input will boot the install/rescue system):
- 'arch <any_other_boot_option>' to boot the install/rescue system.
- 'arch root=/dev/??? <any_other_boot_option>' to boot into an existing system.
- 'lowmem <any_other_boot_option>' to boot the 96MB RAM install system.
- 'lowmem root=/dev/??? <any_other_boot_option>' to boot into an existing
  96MB RAM system.
- If you have trouble with IDE drives, use the "ide-legacy" boot option.
- If your system hangs during the boot process, any combinations of the
  boot options noapic acpi=off pci=routeirq nosmp may be useful.
- 'memtest' to start the memory test program memtest86+.


boot: _
```

Archlinux 提供几种不同的引导选项，你可根据自己需要选择。一般情况下，按回车即可。

稍等片刻，Archlinux 引导进入这个画面

```
Virtual consoles 1-4 are active.
--------------------------------
To change virtual console use ALT + F(1-5 or 12)

Logging:
  - vc5 is used for setup logging.
  - vc12 is used for kernel logging.
Documentation:
  - Documentation can be read by executing:
    zcat /arch/archdoc.txt.gz | less
Keymap:
  - To change to a non-US keymap, type 'km' at the console.
Normal Setup:
  - When you are ready, please run '/arch/setup' to install Arch Linux.
For Experts:
  - Use '/arch/quickinst' to install and bypass the setup routine.
Hit ENTER to enter the bash shell ...

[Arch Linux: /]# /arch/setup_
```

该画面包括一些有用的安装信息，如安装日志的记录、文档的查看、键盘映射的更改等。我们直接在命令提示符后输入下列指令，启动 Archlinux 安装程序：

```
/arch/setup
```

作业平台

`/arch/setup` 是在安装介质中 FreeBSD 风格的安装脚本，安装 Archlinux ，其实就是进入作业平台后，运行这个脚本。

使用 Archlinux 安装光盘引导，就是进入作业平台的过程。作业平台通常包含：

内核

作业平台需要可以使用其它工具，正在运行着的系统是起码的条件，这就需要有内核

磁盘工具

将系统安装到磁盘上，必然要对分区进行相关操作

网络工具

现代 Linux 系统大都支持通过网络安装，各种网络工具也是必需的

包管理系统

在安装过程中，如果希望选择软件，通常需要包管理系统

否则就只能将预先打包的整个系统一古脑的恢复到机器上，且不能选择软件，如 Ubuntu LiveCD

编译工具链

如果采用编译的方式安装，则需要编译工具链，如 Gentoo LFS

基本工具集

以上组件可能会依赖其它的工具，而且安装光盘通常也被当作系统维护光盘，所以基本工具集是必需的

`/arch/setup` 脚本(安装程序)将在“字符图形”界面中调用上述工具:

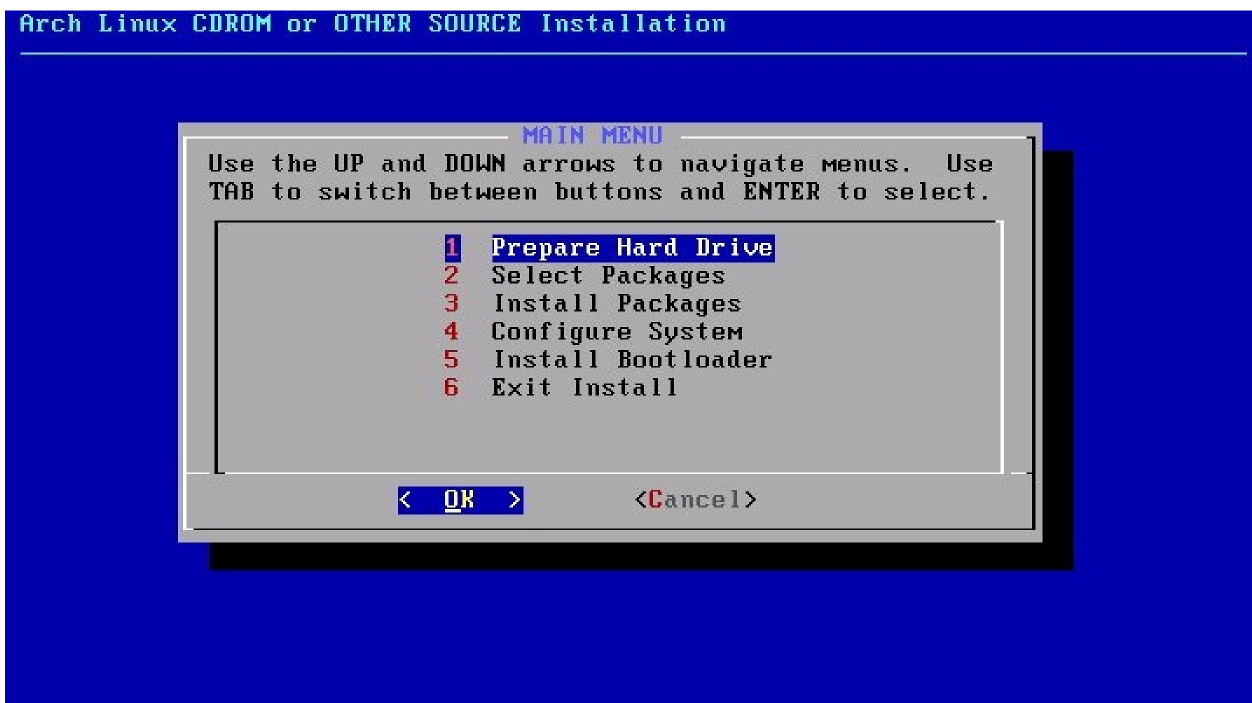
选择安装源



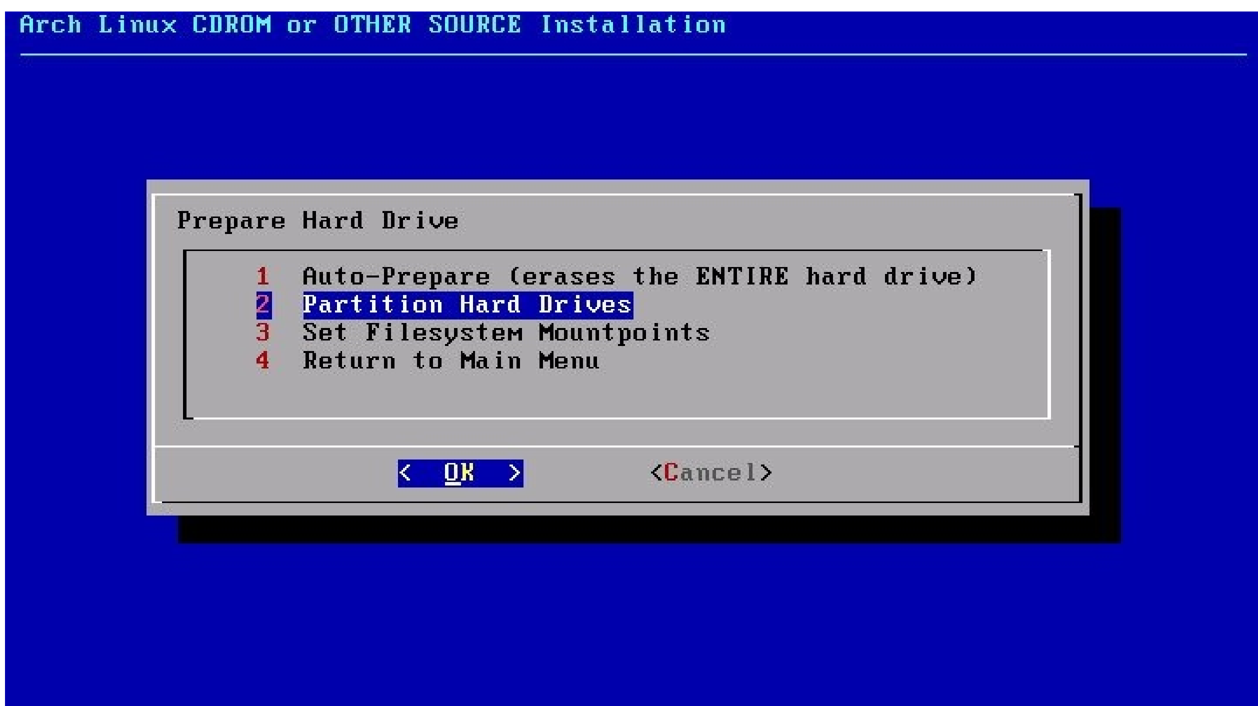
如果使用已包含基本系统软件包的 CORE ，使用默认选项即可；如果通过 FTP/HTTP 下载软件包安装，可以选择第二项，安装程序会自动调用网络工具。

分割磁盘

接下来进入 Archlinux 安装主菜单。



现在需要准备硬盘，即对硬盘分区和挂载文件系统，相关基础概念请参阅 “分区概念”一节



- 自动分区(强烈不推荐)
- 对硬盘分区
- 设置挂载点
- 返回主菜单

安装程序会自动检测硬盘类型及容量，要求你选择硬盘，并调用硬盘分区程序 cfdisk



- 通过“上下方向键”选择分区，“左右方向键”、“TAB”或“首字母大写”切换操作选项
- New 创建分区，需要在 “Free Space”上操作
- Bootable 设置可引导标志
- Write 将分区方案写入分区表

推荐为以下目录建立分区：

```
/
```

根目录。系统将安装在这里，通常 5～10G 足够

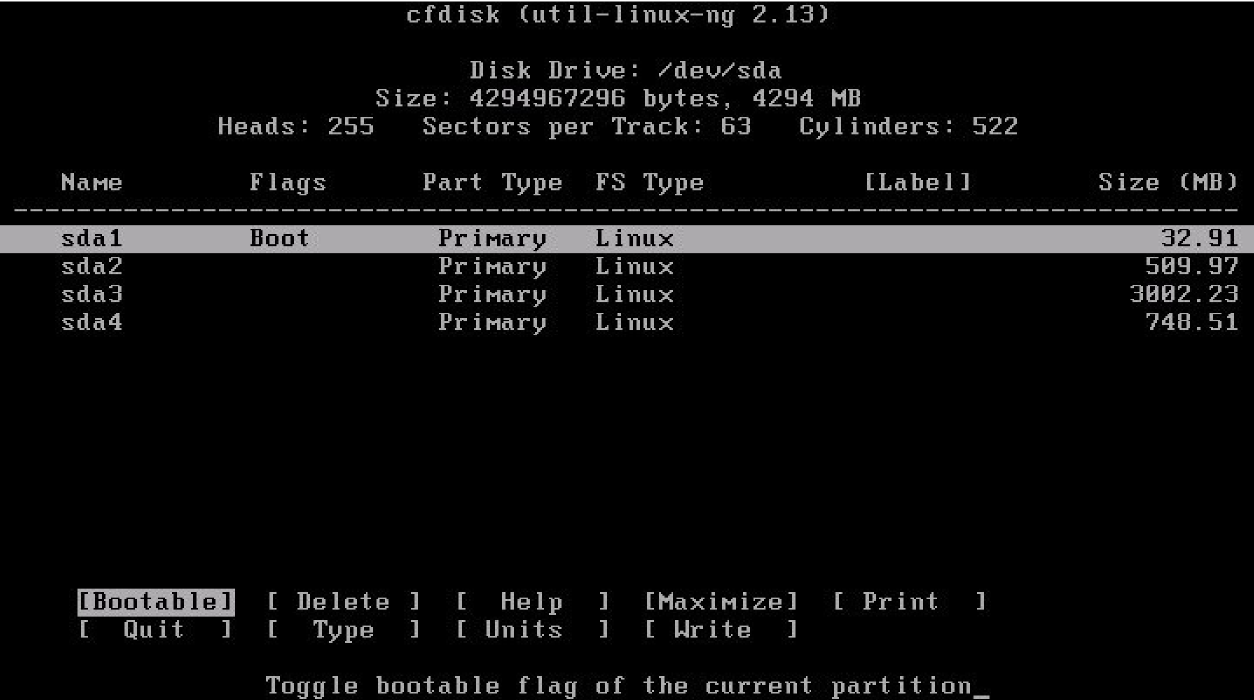
```
/home
```

用户家目录。用户的所有文件都在这里，尽可能的大

```
swap
```

交换分区。物理内存的1～2 倍(如果内存足够大，也可以不建立此分区)

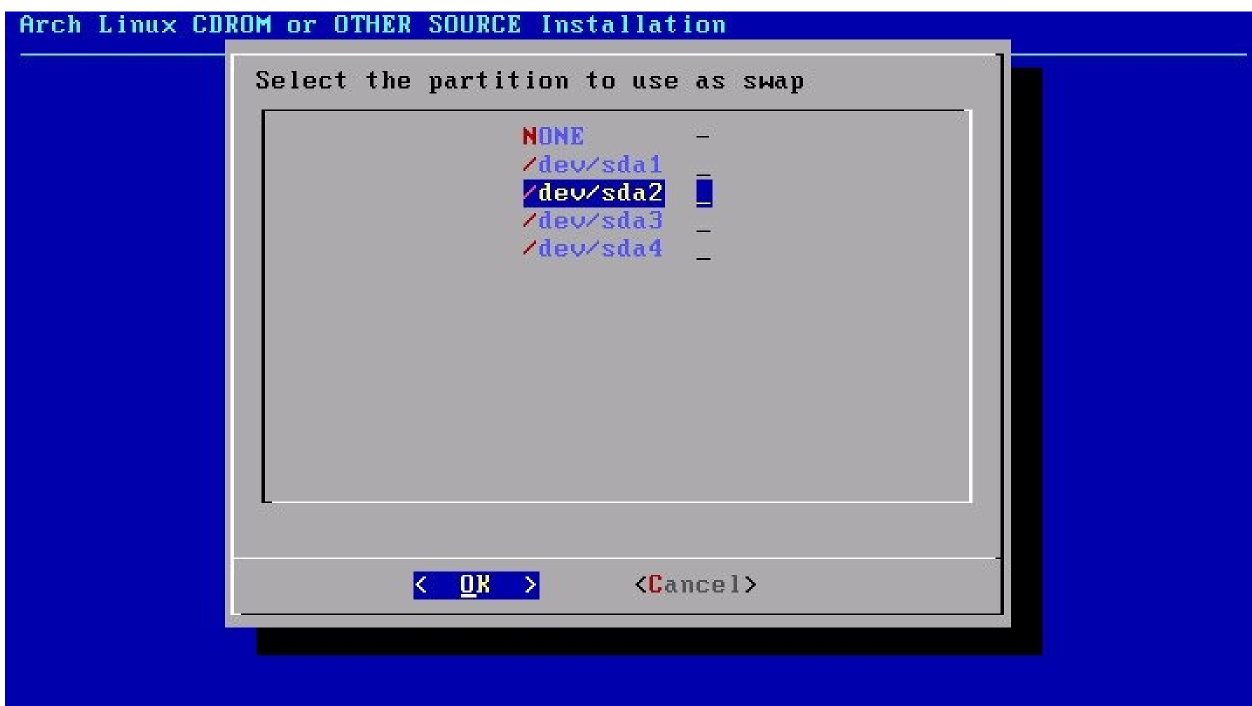
建立好分区后，将根目录所在的分区设置为 Bootable[76]



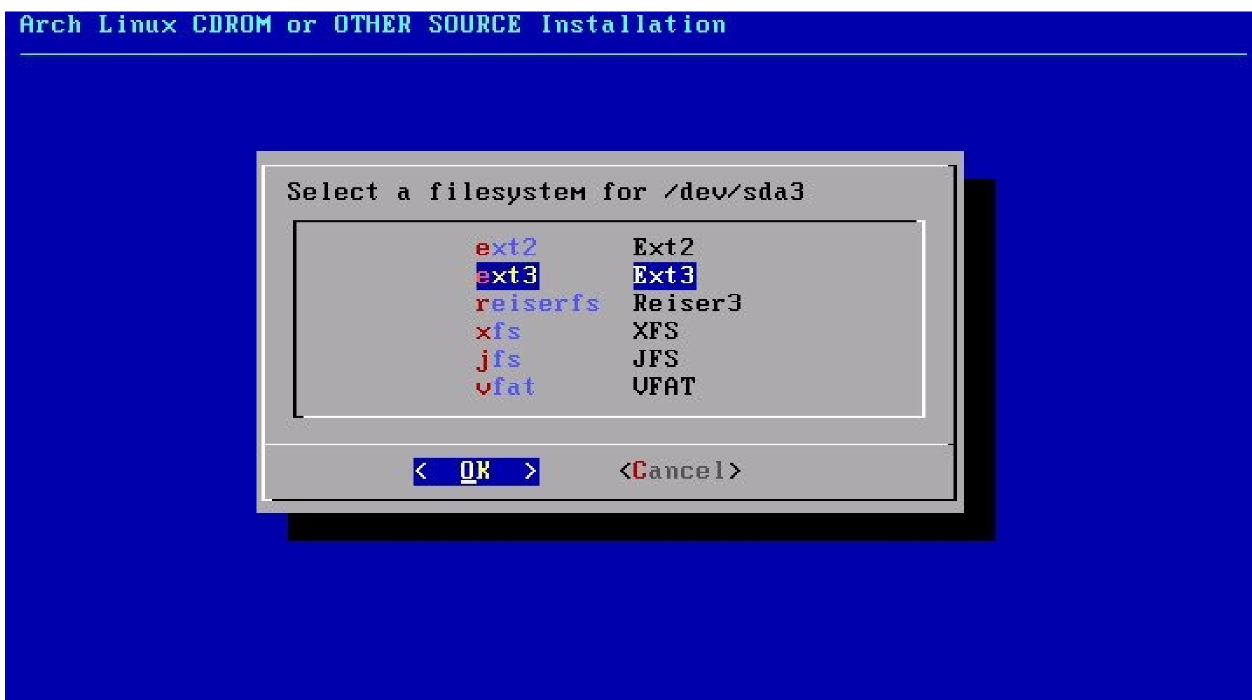
准备妥当后，按下 `W` 写入分区表，`Q` 退出

挂载分区

分割磁盘后，选择 DONE 返回准备硬盘菜单。接着，我们选择 Set Filesystem Mountpoints 来将分区挂载到文件系统上。



首先要求你挂载 swap ，我们选择 /dev/sda2。然后依次挂载 / 和 /home 目录。与挂载 swap 区不同的是，挂载后几个分区会要求你选择文件系统类型，推荐选择 XFS[77]。另外，挂载 /home 时，需要自己输入挂载点，按原名输入即可。



完成后，返回安装程序主菜单。

选择软件

接下来选择要安装的软件包。Archlinux 首先会要求选择安装介质，因为我们是从 CD-ROM 进行安装，所以保持默认。然后，我们选择 CD 驱动器，仍然默认。最后，选择软件包

软件包有四大类：



base

最基本的包

devel

包含一些软件编译工具

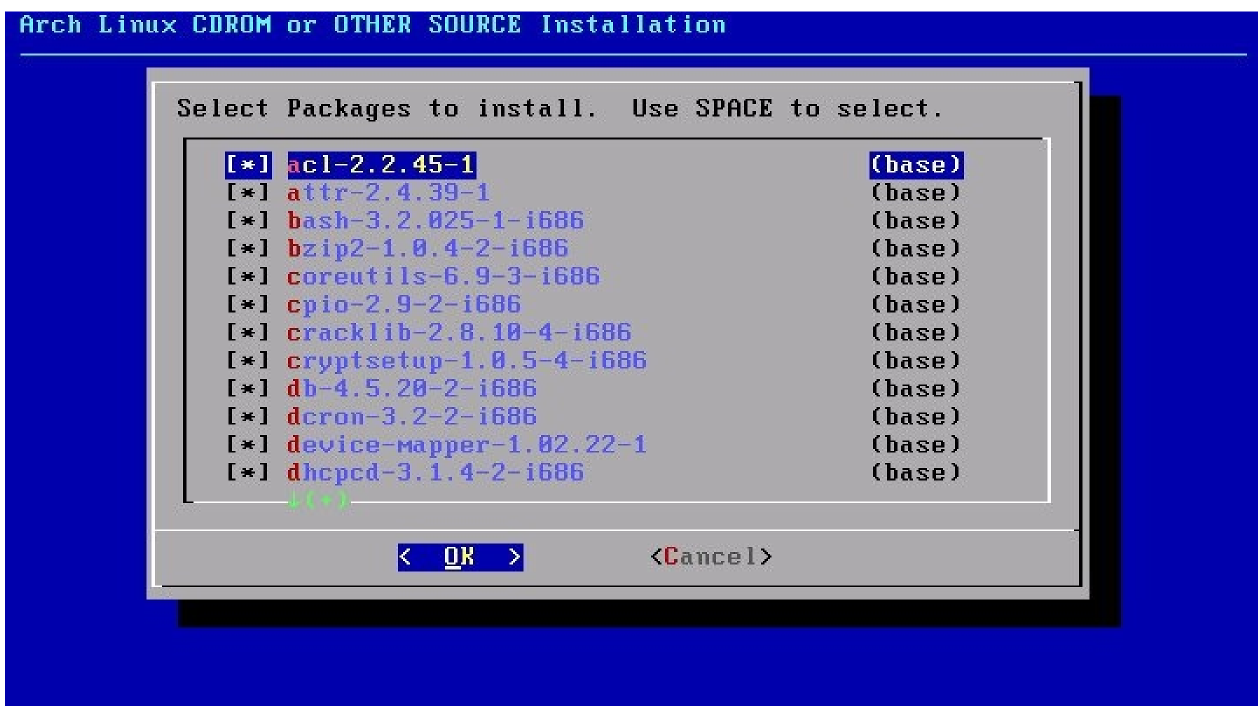
lib

包含应用程序所需的库文件

support

包含一些在网络和文件系统方面有用的包

使用空格键 标记/取消标记。当 Archlinux 安装程序提示你是否默认选中所有的包时，按 Yes 后会进入已标记分类包含软件的选择菜单，完成后按 OK 确认。

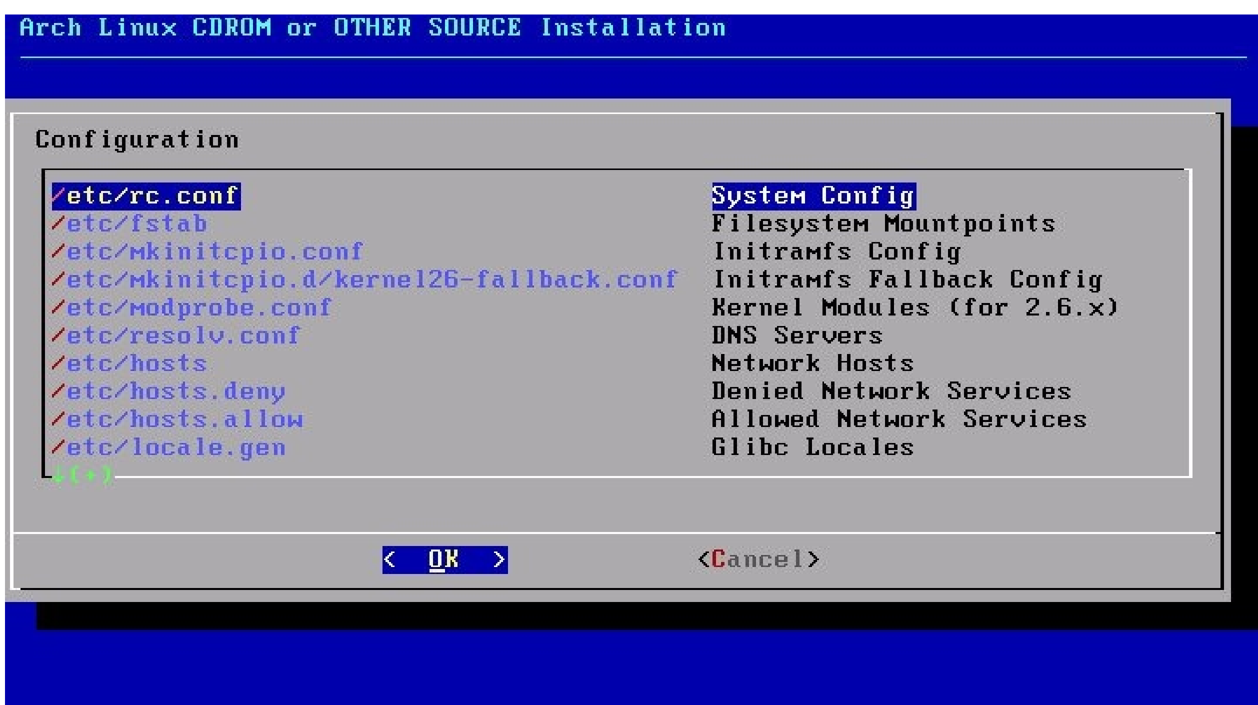


在开始安装前，安装程序将提示你是否保存 Pacman 缓存的软件包，我们选择 No。安装将调用 Pacman 安装你所选择的包，这个过程需要花一会时间。

配置系统

现在，我们将进入 Archlinux 安装过程中的一个重要环节，配置系统文件。Archlinux 安装程序先会询问是否使用 hwdetect，按推荐选择 Yes，并回答是否需要支持从 usb、firewire、pcmcia 等设备引导。

之后，我们需要选择文本编辑器，可选 nano 和 vim，选择后者。然后，我们就到了如下的配置画面：



配置的过程就是调用 Vim[78] 编辑器编辑以上配置文件。其中，需要重点关注的配置文件是：

/etc/rc.conf

/etc/rc.conf 是 Archlinux 最重要的配置文件。下面的选项有必要设置一下：

推荐值	可选值	说明
LOCALE="en_US.utf8"	zh_CN.utf8	系统语言
TIMEZONE="Asia/Shanghai"	UTC[a]	时区
HOSTNAME="myhost"	linuxtoy.org	主机名
MODULES=()	!pcspkr	禁用扬声器
INTERFACES=(eth0)	网络接口	
eth0="dhcp"	eth0 IP地址 netmask 子网掩码 broadcast 广播地址(该IP段最后一个地址“255”)	接口地址设置
DAEMONS=(syslog-ng !network netfs crond)	守护进程以 `!` 起始表示禁用，以 `@` 起始表示后台运行	启动时运行的守护进程

[a] 如果安装了 Windows 系统，使用 UTC 可以避免各个系统时间不一致

/etc/locale.gen

这个文件包含系统所支持的区域及字符集。对中文用户来说，你需要去掉包括 zh_CN 这几行行首的注释符 `#`

```
#ve_ZA  UTF-8
#vi_VN.TCVN     TCVN5712-1
#vi_VN  UTF-8
#wa_BE  ISO-8859-1
#wa_BE@euro     ISO-8859-15
#wa_BE.UTF-8    UTF-8
#xh_ZA.UTF-8    UTF-8
#xh_ZA  ISO-8859-1
#yi_US.UTF-8    UTF-8
#yi_US  CP1255
zh_CN.GB18030   GB18030
zh_CN.GBK       GBK
zh_CN.UTF-8     UTF-8
zh_CN   GB2312
#zh_HK.UTF-8    UTF-8
#zh_HK  BIG5-HKSCS
#zh_SG.UTF-8    UTF-8
#zh_SG.GBK      GBK
#zh_SG  GB2312
#zh_TW.EUC-TW   EUC-TW
#zh_TW.UTF-8    UTF-8
#zh_TW  BIG5
#zu_ZA.UTF-8    UTF-8
#zu_ZA  ISO-8859-1
                                                  387,1          Bot
```

/etc/fstab

该文件确定文件系统设置及挂载点，可以不用编辑，不过查看一下是否正确还是有必要的。

```
#
# /etc/fstab: static file system information
#
# <file system>        <dir>         <type>     <options>            <dump> <pass>
none                   /dev/pts      devpts     defaults              0      0
none                   /dev/shm      tmpfs      defaults              0      0

/dev/cdrom /mnt/cdrom   iso9660   ro,user,noauto,unhide   0      0
/dev/fd0 /mnt/fd0   vfat   user,noauto   0      0
/dev/sda1 /boot ext2 defaults 0 1
/dev/sda2 swap swap defaults 0 0
/dev/sda3 / ext3 defaults 0 1
/dev/sda4 /home ext3 defaults 0 1
~
~
~
~
~
~
~
~
~
~
"/mnt/etc/fstab" 14L, 532C                                 1,1           All
```

设置 **root** 密码

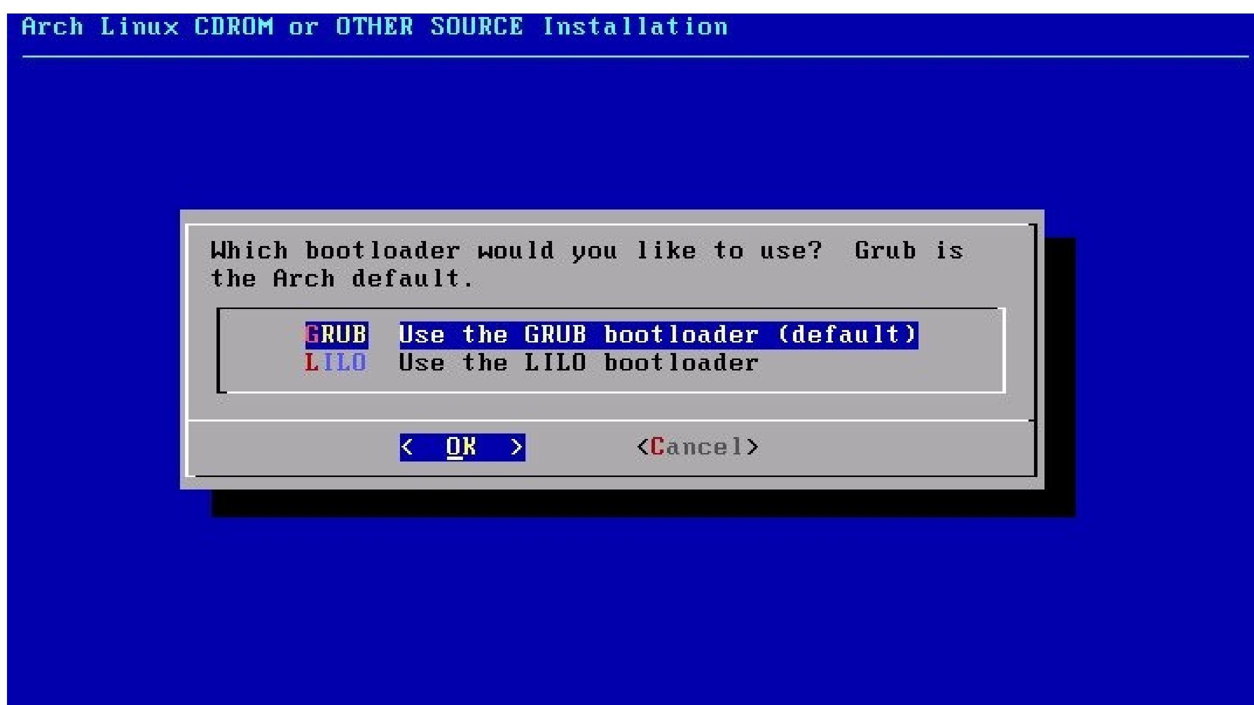
为 root 帐号设置一个密码，需要重复输入确认。

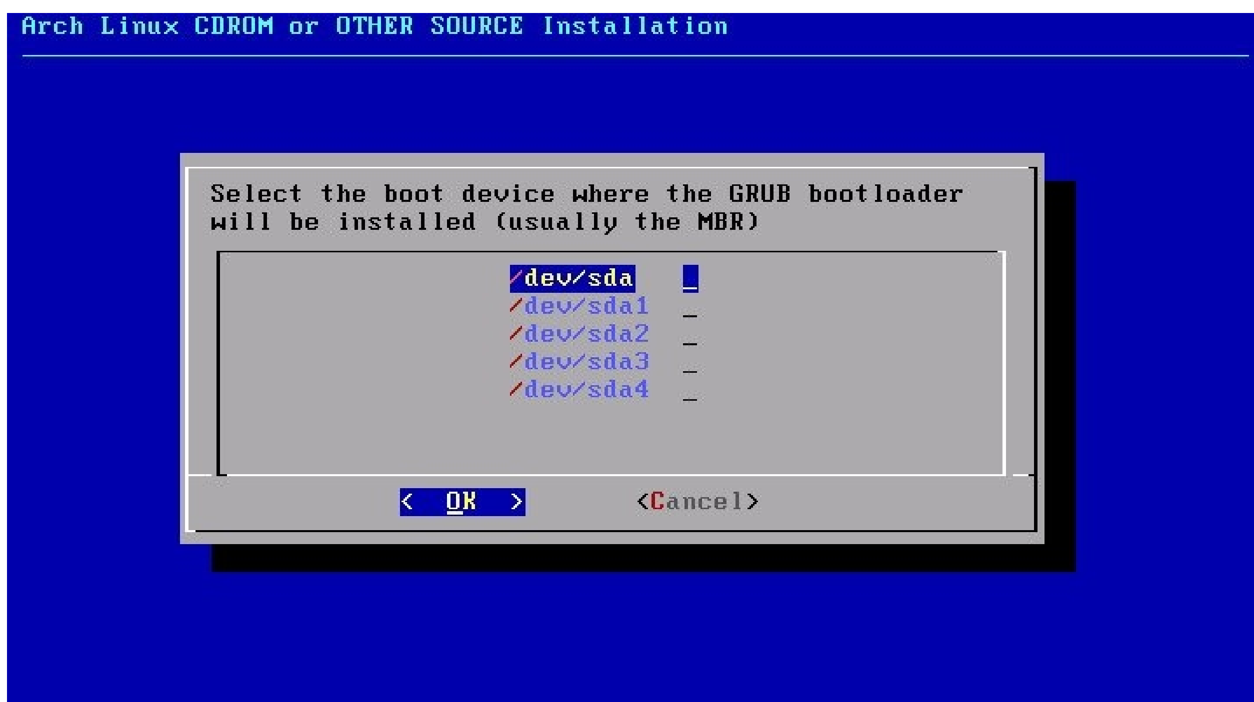
设置 **Pacman** 仓库镜像

为 Pacman 包管理系统设置仓库镜像，选择一个较快的地址。

安装引导程序

系统配置完成后，回到主菜单。进入下一步，安装系统引导程序。我们选择 GRUB 。此时，安装程序让你查看 /boot/grub/menu.lst 的内容。接着，要求选择安装的位置，我们选 MBR，主引导记录



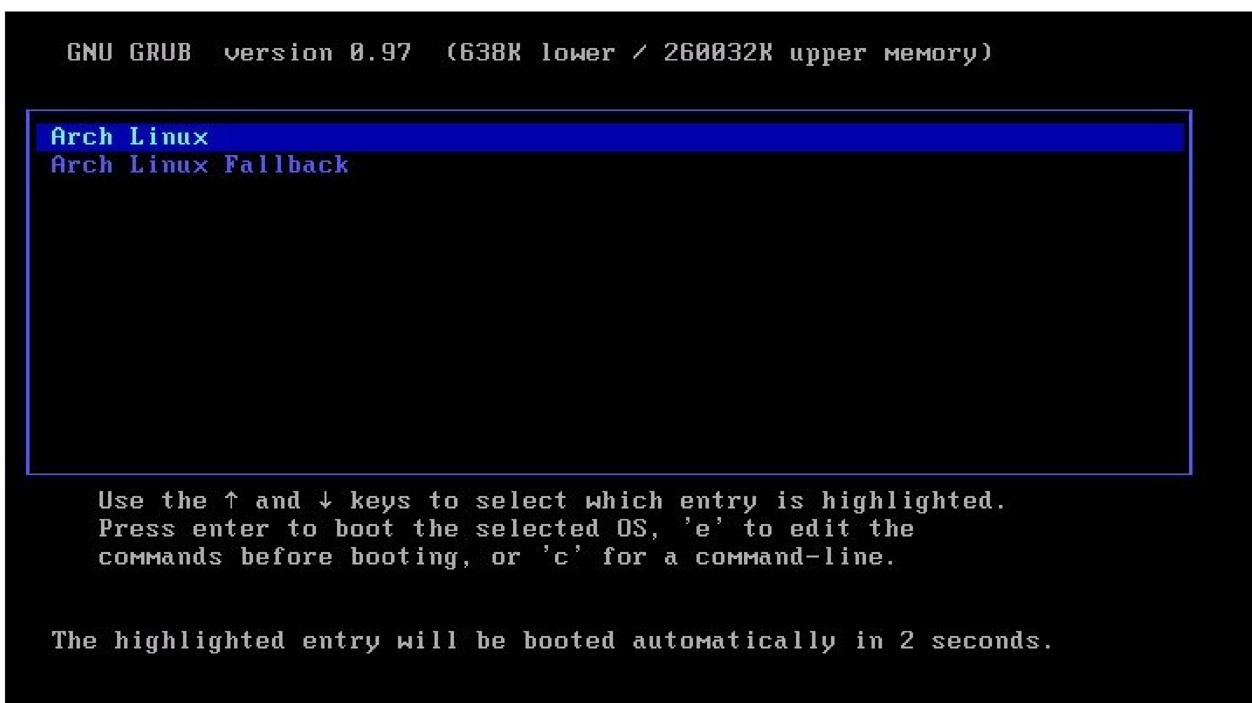


退出安装

如果一切顺利，你已经完成了 Archlinux 的安装。现在是时候退出了。按提示我们在命令行输入下列指令，以便重启系统：

```
reboot
```

新建用户



重启系统后，输入 root 帐号和密码登录系统。



第一件事，建立一个普通的帐号。可以通过以下命令完成：

```
useradd -m -s /bin/bash kardinal
```

这将添加一个名为 kardinal 的用户。接着，为该账号设置密码：

```
passwd kardinal
```

使用 **visudo** 命令将该账号加入 sudoers 列表

确认无误后，可以锁定 root 账号，以绝后患

```
su kardinal      #切换到普通账号，如果能够锁定，说明这个账号是 sudoers
sudo usermod -L root
```

至此，Archlinux 基本系统安装完成

> [75] 将 CORE 中的软件包移除，差不多就是 FTP Install
>
> [76] 如果为 `/boot` 目录准备了独立的分区，则要将该分区设置为 Bootable
>
> [77] 大多数 Linux 系统中的引导程序 grub，不支持从 XFS 分区启动，解决的办法是将 `/boot` 目录挂载到一个独立的 ext2 分区……不过 Archlinux 中的 grub 不存在这个问题
>
> 传说 ext4 文件系统也十分优秀，不过大多数发行版还没有正式支持
>
> [78] 参阅第 24 章 *Vim 编辑器*

更新系统

通过路由器连接到网络，只要设置 `/etc/rc.conf` 文件中的 `eth0` 字段为 dhcp 或者 固定 IP。如果通过 ADSL(PPPoE) 拨号上网，则要执行 **pppoe-setup** 脚本，设置 ADSL 连接，然后通过以下命令连接/断开网络：

```
/etc/rc.d/adsl start
/etc/rc.d/adsl stop
```

更新 Archlinux 系统 (需要 root 权限，如果是普通用户，可以 sudo 执行或者 su 切换用户)：

```
pacman -Syu
```

安装 X.Org

X.Org 是 X Window 的开源实现。如果我们要在 Archlinux 中运行图形化的程序，那么 X.Org 是必不可少的。执行命令：

```
pacman -S xorg
```

该指令将为你安装 X.Org 所必需的包，包括 X.Org 服务器、工具、字体、键盘驱动、鼠标驱动、显卡驱动等等

需要注意的是，默认的 X.Org 安装可能并没有包含你的显卡驱动程序。因此，你需要单独为你的显卡安装驱动。你可以通过下列命令来进行搜索：

```
pacman -Ss xf86-video        #搜索相关驱动
pacman -S xf86-video-vmware  #安装 VMware 驱动
pacman -S nvidia             #安装 nVIDIA 驱动
pacman -S nvidia-96xx        #安装旧版 nVIDIA 驱动
pacman -S nvidia-71xx
```

X.Org 对于即插即用的支持越来越好，不过有时还是要使用 xorgconfig 工具来生成 `xorg.conf` 文件。xorgconfig 是一个交互式的程序，它会向你问一些有关鼠标、键盘、显示器、显卡等方面的问题。根据你的实际情况回答即可。

如果需要，可以手工调整 X.Org 配置文件，参见"X服务器"一节

安装桌面环境

登录管理器

我们选择 GDM 作为登录管理器

```
pacman -S gdm
```

安装完成后，编辑 `/etc/rc.conf` ，在 DAEMONS 字段中添加 gdm

```
DAEMONS=(syslog-ng network netfs crond gdm)
```

Xfce 桌面环境

如何选择桌面环境，是一个倍受争议话题。对于新手，我推荐 Xfce ，它有相对玲珑的体形和丰富的功能。

```
pacman -S xfce4
```

Shell

Zsh 十分强大和人性化，推荐

```
pacman -S zsh
```

终端

在 Linux 中工作，未必一定需要命令行，不过如果你通过读这本书来学 Linux，那么你一定得安个称手的终端：

```
pacman -S rxvt-unicode
```

中文字体

通过以下命令安装中文字体

```
pacman -S ttf-arphic-uming ttf-arphic-ukai
```

不过这两种字体恐怕不能取悦大家挑剔的眼光。你可以自己获取心仪的字体并安装到系统，参阅“XFT字体”一节

中文输入法

在 Archlinux 中包含 SCIM 和 Fcitx 中文输入法。我们选择安装后者：

```
pacman -S fcitx
```

当 Fcitx 安装完成后，将下列内容添加到你用户主目录中的 `.profile` 中，以便让 Fcitx 自动启动：

```
export XMODIFIERS=@im=fcitx
export GTK_IM_MODULE=xim
export QT_IM_MODULE=xim
fcitx &
```

关于中文环境的设置，可以参阅“locale 策略”一节

ALSA

首先确定一下，你需要声音么？你要是想要的话你就说话嘛，你不说我怎么知道你想要呢，虽然你很有诚意地看着我，可是你还是要跟我说你想要的。你眞的想要吗？那你就拿去吧！你不是眞的想要吧？难道你眞的想要吗？

看来你眞的想要，实际上，ALSA 已被包含到 2.6 版的内核中。因此，我们只需安装 ALSA 的相关工具即可

```
pacman -S alsa-utils
```

接着，我们将当前用户添加到 audio 用户组，以便使用声卡设备 (注意把 kardinal 换成你的用户名)：

```
gpasswd -a kardinal audio
```

同时，将 alsa 添加到 `/etc/rc.conf` 配置文件的 DAEMONS 中：

```
DAEMONS=(syslog-ng network netfs crond gdm alsa)
```

ALSA 默认是静音状态，你需要先打开音量：

```
sudo alsaconf #可能你需要配置一下先
alsamixer     #使用“左右方向键”选择，使用“上下方向键”调节，ESC 键退出
```

应用软件

网络浏览

Firefox 几乎是 Linux 平台的标准装备

```
pacman -S firefox firefox-i18n
```

- firefox-i18n 为语言包，安装后可以使 Firefox 界面显示中文，不安也可以正常显示中文页面

Opera 使用 QT 图形库，推荐 KDE 桌面环境下使用

```
pacman -S opera
```

下载工具

命令行下载工具有 wget。此外，aria2 也不错，它支持断点续传和多线程下载：

```
pacman -S aria2
```

BitTorrent 下载工具，我们选用 Deluge：

```
pacman -S deluge
```

其他的包括 Azureus、rTorrent 等。

另外，我们把 aMule 也安装上：

```
pacman -S amule
```

为了能够让 aMule 直接从 Firefox 浏览器中处理 ed2k 链接，我们在 Firefox 的 `about:config` 中新建字符串 `network.protocol-handler.app.ed2k` ，并将其设为 `/usr/bin/ed2k` 。

办公处理

“所见即所得”的办公套件，首推 OpenOffice.org

```
pacman -S openoffice-base openoffice-zh_cn
```

为了让 OpenOffice.org 运行于 GTK 2 模式，我们向 ~/.bashrc 添加如下内容：

```
export OOO_FORCE_DESKTOP=gnome
```

图像编辑

图像编辑软件首选 GIMP，要安装它可以执行命令：

```
pacman -S gimp
```

矢量图形编辑软件可以使用 Inkscape：

```
pacman -S inkscape
```

用 Scrot 截图：

```
pacman -S scrot
```

即时通讯

要与朋友即时聊天，我们可以选用 Pidgin，它支持 Gtalk、MSN、QQ 等多种协议：

```
pacman -S pidgin
```

如果安装 QQ for Linux ，只要下载 tar 包，解压后运行即可

音影播放

音乐播放软件我选择 Quod Libet，你可以凭自己的喜好来安装：

```
pacman -S quodlibet
```

其他的音乐播放器包括 MPD、Audacious、Exaile、Amarok 等。

如果是看电影的话，MPlayer 不错，同时也加上浏览器插件和常用解码器：

```
pacman -S mplayer mplayer-plugin codecs gstreamer0.10-bad gstreamer0.10-ugly \
gstreamer0.10-ffmpeg gstreamer0.10-mad gstreamer0.10-mpeg2dec
```

如果需要 MPlayer 的前端，那么可以安装 SMPlayer：

```
pacman -S smplayer
```

当然，另一个选择 VLC 也挺好：

```
pacman -S vlc
```

新闻阅读

RSS 离线阅读软件，我们安装 Liferea：

```
pacman -S liferea
```

图像查看

GQview 是一个轻快的图像查看软件：

```
pacman -S gqview
```

文本编辑

如果要求简单的话，可以选用 Gedit、Leafpad：

```
pacman -S gedit
pacman -S leafpad
```

Scite 功能强大，操作简单，推荐

```
pacman -S scite
```

Vim 似乎也是 Linux 系统的标准装备，需要图形界面的话，可以加上 Gvim：

```
pacman -S vim
```

Emacs 的最新版本为 emacs-cvs

```
pacman -S emacs-cvs
```

如果只是喜欢 Emacs 的操作方式，可以安装轻量级的 类Emacs 编辑器 Jed

```
pacman -S jed
```

FTP 客户端

Lftp 是命令行的 ftp 客户端，但是它很、十分、非常、以及特别的好用

```
pacman -S lftp
```

Lftp 的使用方法参见第 42 章 使用 *lftp*

图形化的有 gFTP、FileZilla 等。

光盘刻录

我们选择 K3b，你可以通过以下指令安装：

```
pacman -S k3b
```

为了让当前用户能够使用光盘刻录设备，需要将其添加到 optical 用户组中 (请将 kardinal 替换成你的用户名)：

```
gpasswd -a kardinal optical
```

文档查看

查看 PDF 文档，可以安装 Evince：

```
pacman -S evince
```

CHM 文档，可以选用 KchmViewer。

其它工具

计算器：

```
pacman -S gcalctool
```

压缩/解压 rar、zip 等格式：

```
pacman -S unrar unzip
```

另外，图形化的可以用 Squeeze：

```
pacman -S squeeze
```

Flash 插件：

```
pacman -S flashplugin
```

Java 支持：

```
pacman -S jre
```

词典翻译，我们安装 StarDict：

```
pacman -S stardict
```

词典文件需从 StarDict 官方网站 下载，并释放到 `/usr/share/stardict/dic/` 目录。

Compiz Fusion

Compiz Fusion 合并自 Compiz 和 Beryl，它不仅将 Linux 桌面带入了 3D 环境，而且包含许多既丰富又渲丽的效果。

准备配置文件

要在 Archlinux 中安装 Compiz Fusion，首先确保 `/etc/X11/xorg.conf` 文件的正确配置。以 NVIDIA 显卡为例：

```
#Module 部分载入 GLX 模块：
Load "glx"

#Device 部分添加下列选项：
Option "AddARGBGLXVisuals" "True"
#以上是针对较新卡的配置，如果是使用旧卡的话，那么还应加上：
Option "RenderAccel" "true"
Option "AllowGLXWithComposite" "True"

#添加 Extensions 部分：
Section "Extensions"
Option "Composite" "Enable"
EndSection
```

安装 Compiz Fusion

现在，让我们来安装 Compiz Fusion，执行下列命令：

```
pacman -S compiz-fusion
```

这将安装 Compiz Core、Compiz Fusion 插件、Compiz Fusion 设置管理器、Emerald 及主题、Fusion Icon 等。

另外，GNOME 用户可以安装窗口装饰：

```
pacman -S compiz-fusion-gtk
```

KDE 用户为：

```
pacman -S compiz-fusion-kde
```

自动启动 Compiz Fusion

要启动 Compiz Fusion，可以运行 Fusion Icon，它是一个系统托盘程序，通过它可以很方便的切换：

```
fusion-icon
```

从 Fusion Icon 中，我们可以将窗口管理器切换为 Compiz，窗口装饰切换为 Emerald。此外，该工具也可以调用 Compiz Fusion 设置管理器及 Emerald 主题管理器。具体的调整过程，你不妨亲自试试。

如果打算让 Compiz Fusion 自动启动，可以将 Fusion Icon 加入 GNOME 会话的启动程序组中。方法是，点击“系统 → 首选项 → 会话”，在启动程序标签中点击“添加”按钮，然后输入下列信息：

字段	值
名称	Compiz Fusion
命令	fusion-icon
注释	Compiz Fusion

Avant Window Navigator

Avant Window Navigator 是一个漂亮的 Dock 程序，提供程序启动、窗口管理等，并包含许多插件。

你可以使用下列命令来安装 AWN：

```
pacman -S avant-window-navigator
```

AWN 可通过“应用程序 → 附件 → Avant Window Navigator”启动。自动启动的设置可以参考 Compiz Fusion 的做法。

Pacman

Pacman 是 Archlinux 默认的包管理工具，由 Archlinux 的创始人 Judd Vinet 开发。Pacman 可以很好的处理依赖关系、通过网络使用软件仓库。使用 Pacman，你不仅可以更新 Archlinux 的整个系统，而且能够对包进行管理，包括安装、删除、升级等。同时，Pacman 也允许你搜索包和查看有关包的信息。

配置

Pacman 的配置文件为 `/etc/pacman.conf`，可以在其中设定使用的软件源分支

```
[core]
# 在这里添加你的首选服务器，它们将被优先使用
Include = /etc/pacman.d/mirrorlist

[extra]
# 在这里添加你的首选服务器，它们将被优先使用
Include = /etc/pacman.d/mirrorlist

[community]
# 在这里添加你的首选服务器，它们将被优先使用
Include = /etc/pacman.d/mirrorlist

#[testing]
#Include = /etc/pacman.d/mirrorlist

# 自定义软件仓库的示例
#[custom]
#Server = file:///home/custompkgs

# archlinuxfr 软件仓库
[archlinuxfr]
Server = http://repo.archlinux.fr/i686
```

指定 Pacman 使用的下载工具

```
# 默认(不指定的情况下)为 wget
#XferCommand = /usr/bin/wget --passive-ftp -c -O %o %u
# 使用 aria2 下载 ，删除下一行行首注释符 `#`
#XferCommand = aria2c -s 5 -m 5 -d / -o %o %u
# 使用 curl 下载
#XferCommand = /usr/bin/curl %u > %o
```

命令

Pacman 是一个命令行工具，这意味着当你执行下面的命令时，必须在终端或控制台中进行。

更新系统

在 Archlinux 中，使用一条命令即可对整个系统进行更新：

```
pacman -Syu
```

如果你已经使用 **pacman -Sy** 将本地的包数据库与远程的仓库进行了同步，也可以只执行：

```
pacman -Su
```

安装软件包

这个命令你应该见过无数次了：

```
pacman -S 软件包名称
```

- 如果同时安装多个包，用空格分隔包名

其它用法：

```
# 先同步包数据库再安装
pacman -Sy 软件包名称
# 显示一些操作信息后执行安装
pacman -Sv 软件包名称
# 安装本地软件包，其扩展名为 pkg.tar.gz
pacman -U 软件包名称
```

删除软件包

```
# 只删除软件包，不删除该软件包的依赖
pacman -R 软件包名称
# 删除软件包的同时，也将删除其依赖
pacman -Rs 软件包名称
# 删除软件包、依赖关系、配置文件
pacman -Rsn 软件包名称
# 删除包时不检查依赖
pacman -Rd 软件包名称
```

搜索

通过关键字搜索软件包

```
pacman -Ss `关键字`
```

搜索已安装的包

```
# 查看软件包信息
pacman -Qi 软件包名称
# 列出软件包的文件
pacman -Ql 软件包名称
# 查看某一文件属于哪个软件包
pacman -Qo 文件名
```

假如想知道某一程序的相关信息，可以配合 **whereis** 使用 pacman

```
   `whereis sudo`
sudo: /usr/bin/sudo /usr/share/man/man8/sudo.8.gz

   `pacman -Qo /usr/bin/sudo`
/usr/bin/sudo is owned by sudo 1.6.9p18-1

   `pacman -Qi sudo`
Name : sudo
Version : 1.6.9p18-1
URL : http://www.sudo.ws/sudo/
Licenses : custom  ISC
Groups : None
Provides : None
Depends On : glibc  pam
Optional Deps : None
Required By : None
Conflicts With : None
Replaces : None
Installed Size : 308.00 K
Packager : Allan McRae <allan@archlinux.org>
Architecture : i686
Build Date : Sat 15 Nov 2008 06:17:33 AM CST
Install Date : Fri 21 Nov 2008 12:20:07 PM CST
Install Reason : Explicitly installed
Install Script : No
Description : Give certain users the ability to run some commands as root

   `pacman -Ql sudo`
sudo /etc/
sudo /etc/pam.d/
sudo /etc/pam.d/sudo
sudo /etc/sudoers
sudo /usr/
sudo /usr/bin/
sudo /usr/bin/sudo
sudo /usr/bin/sudoedit
sudo /usr/lib/
sudo /usr/lib/sudo_noexec.so
sudo /usr/sbin/
sudo /usr/sbin/visudo
sudo /usr/share/
sudo /usr/share/licenses/
sudo /usr/share/licenses/sudo/
sudo /usr/share/licenses/sudo/LICENSE
sudo /usr/share/man/
sudo /usr/share/man/man5/
sudo /usr/share/man/man5/sudoers.5.gz
sudo /usr/share/man/man8/
sudo /usr/share/man/man8/sudo.8.gz
sudo /usr/share/man/man8/sudoedit.8.gz
sudo /usr/share/man/man8/visudo.8.gz
```

其他

```
# 只下载软件包，不安装
pacman -Sw 软件包名称
# Pacman 下载的软件包缓存于 /var/cache/pacman/pkg/ 目录。清理未安装的包
pacman -Sc
# 清理所有缓存的文件
pacman -Scc
# 搜索孤立软件包
pacman -Qdt
```

编译系统

备份、恢复与迁移

备份 Linux 系统，推荐使用 tar。使用 archlinux-2008.06-core-i686 光盘启动系统[79]，登录“作业平台”后，首先挂载文件系统

> 注意：备份、恢复、迁移等过程，都要先进行这一步

```
#首先将两个工作目录定义为环境变量，这样条理更清晰
export FROM="/mnt/from"   #定义变量 `FROM`，待备份的文件系统挂载于此
export TO="/mnt/to"       #定义变量 `TO`，备份文件存放路径
mkdir -p $FROM $TO        #创建工作目录
#假设待备份的系统位于 /dev/sda1 分区
mount /dev/sda1 $FROM
#如果将系统中的目录挂载到其它分区，例如将 /boot 挂载于 /dev/sda2 ，还要继续挂载此目录
mkdir $FROM/boot
mount /dev/sda2 $FROM/boot
mkdir $FROM/home
mount /dev/sda3 $FROM/home
……
#假设将备份文件存放于 /dev/sda5 分区
mount /dev/sda5 $TO
```

使用 **tar**(参见“压缩解压”一节) 命令备份：

```
cd $FROM
tar -zcvf $TO/backup.tgz          \
        --exclude=backup.tgz      \
        --exclude=mnt/*           \
        --exclude=proc/*          \
        --exclude=sys/*           \
    *
```

系统更新后，可以使用以下命令，在原来备份的基础上进行差异备份

```
#查找最近改动的文件，生成文件列表
find $FROM -mtime -1 -print> filelist
#根据文件列表进行差异备份
tar -zcv -T filelist -f $TO/backup.tgz
```

恢复时，使用以下命令

```
tar zxvpf $TO/backup.tgz -C $FROM
```

在某一硬件架构中，Linux 中大部分文件与硬件无关。所以，将备份的系统迁移到其它机器上[80]，是可行的

恢复后，有三个文件需要重新编辑：`/boot/grum/menu.lst` 、`/etc/fstab` 、`/etc/X11/xorg.conf`

使用 blkid，输出磁盘设备的 UUID，根据实际情况编辑这两个文件

```
blkid >> /etc/fstab
blkid >> /boot/grub/menu.lst
```

使用 `grub` 命令安装引导器，参见“Grub 安装”一节

> [79] 也可以使用其它 LiveCD
>
> [80] 例如将虚拟机中安装的 Linux 系统迁移到真实机器上

第 32 章 组织你的意念：Emacs org mode

目录

引子

眞正优秀的软件，通常都包括多个平台的版本，OneNote 是个例外[81]

不只一次，我听到很多 Linux 的用户抱怨开源软件中没有可以替代 OneNote 的软件。当然也有许多种权宜之计来解这个燃眉之急，比如 SunBird、osmo，甚至是 Wiki、Blog

SunBird、osmo 的操作太过 Windows，并且不如 OneNote 好用；Wiki、Blog 部署成本太高，且不够灵活……

Emacs 的 Org-mode 在一定程度上可以替代 OneNote，有些方面甚至更好

	Org-mode	OneNote
标签	强大	不支持
日程表	强大	不支持
界面	字符	漂亮
TablePC	不支持	非常好
摘录	保持源格式	
便捷	Emacs 内置	安装麻烦

> [81] 不得不承认，OneNote 在 Windows 平台原生软件中罕见优秀

简介

Org-mode 主要包含标签、待办、日程表几大部分

规模效应. 如果资料只有几十几百条，借助分类的方式可以有效管理，但是资料的条目超过了一定的数量，标签更管用。好比传统的邮箱，使用分类的方式管理邮件；而 Gmail 由于起点容量很大，所以提供了标签

组织. 提供了标签进行宏观控制，别外还有日程表作为快速通道

构思与发布. Org-mode 不但可以整理思路，而且拥有比较完善的发布功能

配置

首先对 Org-mode 进行一些简单的配置，在 `.emacs` 文件中写入：

```
(setq org-hide-leading-stars t)
 (define-key global-map "\C-ca" 'org-agenda)
 (setq org-log-done 'time)
```

> ❶ 只高亮显示最后一个代表层级的 *
>
> ❷ `C-c a` 进入日程表
>
> ❸ 给已完成事项打上时间戳。可选 note，附加注释

建立一个 org 文件

新建一个名为 `sandbox.org` 的文件[82]，头部内容如下：

```
#+STARTUP: overview
#+TAGS: { 桌面(d) 服务器(s) }  编辑器(e) 浏览器(f) 多媒体(m) 压缩(z)
#+TAGS:  { @Windows(w)  @Linux(l) }
#+TAGS:  { 糟糕(1) 凑合(2) 不错(3) 很好(4) 极品(5) }
#+SEQ_TODO: TODO(T) WAIT(W) | DONE(D!) CANCELED(C@)
#+COLUMNS: %10ITEM  %10PRIORITY %15TODO %65TAGS
```

> 提示：这里的内容可以随时更改，但是要记得在改后用 `C-c C-c` 刷新设置
>
> ❶ 启动时概览
>
> ❷ 设定标签，括弧中的为标签快捷键（如果没有指定，默认为首字母）。可以设置在多行中
>
> ❸ 花括号为标签组，组中的标签只能选一个
>
> ❹ 设定待办状态。将项设置为 `|` 后面的状态时（DONE CANCELED），会打上 CLOSED 标志
>
> ❺ 设定列视图

大纲

在这个新建的文件中插入下面内容：

```
* 工作
** Emacs
   神之编辑器
*** org-mode
    组织你的意念
** Vim
   编辑器之神
** EmEditor
* 娱乐
** Mplayer
   全能播放器
* 网络
** firefox
** IE
** PureFTPD
* 其它
** WinRAR
```

- `*` 之后有一个空格，一定不能省略
- 每一个 `*` 代表一级分支，`***` 就代表第三级分支

现在你的 Emacs 应该显示这样的内容

❶

其实这就是一个大纲模式，只是 Org-mode 用更醒目的色彩来显示，并且快捷键方便一些

按下 `S-TAB`，会显示概览：

❷

把光标定位在 `* 工作` 这一行，按几下 TAB

❸

> 提示：连续按下 `TAB` 键，会在显示分支、全部显示、全部隐藏几种状态间循环切换，`S-TAB` 作用于全部分支

以下快捷键控制 Org-mode 显示

C-c C-a	全部显示
C-c C-x b	在一个新缓冲区中显示当前分支

当一个 `org` 文件内容很多时，使用 Emacs 的快捷键移动就很没有效率，可以使用 Org-mode 内建的移动键

	向前	向后
同级	C-c C-f	C-c C-b
跨级	C-c C-n	C-c C-p
上一级	C-c C-u	
跳转	C-c C-j	

对分支结构进行修改：

C-RET	加入新的同级标识
M-left	将当前项提升一级
M-right	将当前项降低一级
M-S-left	将当前分支提升一级
M-S-right	将当前分支降低一级
M-S-up	将当前分支向上移动
M-S-down	将当前分支向下移动
C-c C-x C-k	删除当前分支
C-c C-x M-w	复制当前分支
C-c C-x C-y	粘贴分支
C-c C-w	移动当前分支
C-c *	为当前分支加入内容

内容

可以在项标题下随意的插入内容。通过下面方式插入链接：

```
[[链接地址][链接名称]]
```

如果不想定义链接的名称，可以这样：

```
[[链接地址]]
```

如果想修改这个链接，在链接后按下 `backspace`

> [82] Emacs 打开扩展名为 `.org` 的文件时自动进入 Org-mode

标签

使用

对于信息的管理，有分类和标签两种方式。

分类的方式，每一个项只能属于一个分类，但是分类的方法往往不只一种，比如一个软件，从类型上分，可以有编辑器、浏览器等，从评价上分，可以是好、坏；如果使用类型来分类，就不能使用评价来分类；而标签的方式可以很好的解决这个问题

将光标定位在当前项（Emacs）上，按下 `C-c C-c`

❹

- 注意回显区中的内容：[a-z]用快捷键选取 [SPC]清除所有标签 [RET]确认 [TAB]自由输入（不建议） [!]取消组（同一组签标可以多选） [C-c]单选多选切换

使用快捷键 `s` 选中“服务器”，然后再按 `d` 选中桌面，可以看到，两个不能同时选，因为它们是一组标签，只能选一个。再分别按下 `e` 和 `5`,回车确认：

❺

接下来在 `*** org-mode` 上 `C-c C-c`

❻

可以看到，它自动继承了“桌面”、“编辑器”、“极品”三个标签，而它自己拥有“应用”这个标签

定义

在当前文件头部进行定义：

```
#+TAGS: { 桌面(d) 服务器(s) }  编辑器(e) 浏览器(f) 多媒体(m) 压缩(z)
```

每项之间必须用空格分隔，可以在括弧里定义一个快捷键；花括号里的为标签组，只能选择一个

对标签定义进行修改后，要在文件头部按下 `C-c C-c` 刷新[83]

也可以在 Emacs 配置文件 `.emacs` 中进行定义

```
(setq org-tag-alist '(("编辑器" . ?e) ("浏览器" . ?f) ("多媒体" . ?m)))
```

在配置文件中设置的是全局标签，只要是 org-mode，无论是否在头部进行设置，都可以使用这些标签。不推荐这种方式

查询

设置标签的主要目的还是为了查询。org-mode 会为搜索结果建立一个视图

C-c \	搜索标签
C-c / T	
C-u C-c \	搜索带 TODO 的标签

可以使用逻辑表达式限制条件，更准确灵活的搜索

+	和	a+b	同时有这两个标签
-	排除	a-b	有 a 但没有 b
|	或	a|b	有 a 或者有 b
&	和	a&b	同时有 a 和 b，可以用“+”替代

在查询视图中 `C-c C-c` 退出

> [83] 在文件头部按下 `C-c C-c` 为刷新设置；在项标题按下 `C-c C-c` 为设置标签

事件

事件默认有两种状态“TODO”、“DONE”，在项标题上 `C-c C-t` 切换事件状态；`M-S-RET` 新建事件

定义

在文件头部定义事件状态：

```
#+SEQ_TODO: TODO | DONE  CANCELED
#+SEQ_TODO: TODO(T) | DONE(D)  CANCELED(C)
#+SEQ_TODO: TODO(T!) | DONE(D@)  CANCELED(C@/!)
```

> ❶ `|` 分隔完成与未完成两种状态，完成状态会打上 CLOSED 时间戳
>
> ❷ 设定快捷键
>
> ❸ `!` 打上时间戳；`@` 要求说明

也可以在配置文件中设置全局事件状态：

```
     (setq org-todo-keywords
           '((sequence "TODO" "|" "DONE"  "CANCELED")
             (sequence "REPORT" "BUG" "KNOWNCAUSE" "|" "FIXED")
             ))
```

操作

C-u C-c C-t	手动输入 TODO 状态，如果设定快捷则使用快捷键输入
S-right	循环切换 TODO 状态，两个以上 TODO 状态时使用
S-left	
C-S-right	组间切换
C-S-left	
C-c C-v	查询视图
C-c / t	
C-c a t	全局 TODO 列表

日程表

在 Emacs 配置文件 `.emacs` 定义日程表快捷键：

```
(define-key global-map "\C-ca" 'org-agenda)
```

`C-c [` 将当前文件加入日程表，`C-c a`

❼

a	本周事件
t	显示所有事件
m	查询标签
L	当前缓冲区时间线
s	查询关键词
T	查询带 TODO 关键词的项
M	查询带 TODO 关键词的标签
#	显示已停止事件
q	退出日程表

可以将多个 org 文件从日程表加入、移除，或者将日程表锁定在当前 org 文件的某个分支：

C-c [	将当前文件加入日程表。如果已加入，移动到前面
C-c]	将当前文件从日程表中移除
C-c C-x <	锁定到当前树(只显示当前树的事件)
C-c C-x >	解除锁定

优先级

事件有“A”、“b”、“C”三种优先级别，使用 `C-c ,` 手动设定级；使用 `S-up` 和 `S-down` 进行调整

进度

可以给事件设定进度，使用类似 `[66%]` 或者 `[2/3]` 这样的形式

时间

`C-c C-d` 设定截止日期(DEADLINE)；`C-c C-s` 设定计划(SCHEDULED)：

❽

`C-c .`在当前位置插入一个时间戳：

❾

时间标记都会显示在日程表的本周事件中:

❿

C-c .	插入时间戳；如果连续插入两个时间戳，则插入一个时间范围
C-u C-c .	更加精确的时间戳，在日程表中以时间线显示
C-c !	插入时间戳，不在日程表中显示
C-c <	直接插入时间戳（当前日期）
C-c >	查看日历
C-c C-o	访问当前时间戳的日程表
S-left|S-right	以天为单位调整时间戳时间
S-up|S-down	调整光标所在时间单位；如果光标在时间戳之外，调整时间戳类型（是否在日程表中显示）
C-c C-y	计算时间范围长度

列视图

`C-c C-x C-c` 进入列视图；按 `q` 退出：

⓫

在文件头部设置列：

```
#+COLUMNS: %10ITEM  %10PRIORITY %15TODO %65TAGS
```

> ❶ 百分数表示该列所占宽度
>
> ❷ 优先级
>
> ❸ 事件状态
>
> ❹ 标签

C-c C-x C-c	进入列视图
r|g	刷新
q	退出
left|right	在列间移动
S-left|S-right	改变当前列的值
n|p	
1~9,0	用编号选择值
v	查看当前值

典型应用

复制保存为 `.org` 文件，尝试进行一些操作

清单

```
#+STARTUP: overview
#+TAGS:  { 工作(x)  娱乐(y) 杂项(z) }
#+TAGS:  { 编辑器(e) 网络(w) 多媒体(m) 压缩(z) 窗口管理器(c) 系统(s) 终端(t) }
#+TAGS:  { 糟糕(1) 凑合(2) 不错(3) 很好(4) 极品(5) }
#+SEQ_TODO: TODO(T) WAIT(W) | DONE(D!) CANCELED(C@)
#+COLUMNS: %10ITEM  %10PRIORITY %15TODO %65TAGS

* Emacs                                            :工作:编辑器:极品:
* Vim                                              :工作:编辑器:很好:
* EmEditor                                         :工作:编辑器:不错:
* bash                                             :工作:系统:很好:
* zsh                                              :工作:系统:极品:
* rxvt-unicode                                     :工作:系统:很好:
* tilda                                            :工作:系统:很好:
* awesome                                          :杂项:窗口管理器:很好:
* openbox                                          :杂项:窗口管理器:很好:
* xfce4                                            :杂项:窗口管理器:不错:
* firefox                                          :杂项:网络:很好:
* IE                                               :杂项:网络:糟糕:
* lftp                                             :杂项:网络:很好:
* wget                                             :杂项:网络:很好:
* aria2                                            :杂项:网络:很好:
* 迅雷                                             :杂项:网络:极品:
* mplayer                                          :娱乐:多媒体:很好:
* 暴风影音                                         :娱乐:多媒体:不错:
```

日志

```
#+STARTUP: overview
#+TAGS:  { 工作(x)  娱乐(y)  }
#+SEQ_TODO: TODO(T) WAIT(W) | DONE(D!) CANCELED(C@)
#+COLUMNS: %10ITEM  %10PRIORITY %15TODO %65TAGS

* <2008-10-21 二>
** DONE <2008-10-21 二 11:38>                           :工作:
   CLOSED: [2008-10-21 二 11:42]
   do ...........
** DONE <2008-10-21 二 11:45>                           :工作:
   CLOSED: [2008-10-21 二 11:42]
   do ...........
** CANCELED <2008-10-21 二 12:38>                       :娱乐:
   CLOSED: [2008-10-21 二 11:42]
   - State "CANCELED"   [2008-10-21 二 11:43] \\
     太忙了
   do ...........
** DONE <2008-10-21 二 14:50>                           :工作:
   CLOSED: [2008-10-21 二 11:43]
   do ...........
* <2008-10-22 三>
** DONE <2008-10-22 三 11:38>                           :工作:
   CLOSED: [2008-10-21 二 11:44]
   do ...........
** WAIT <2008-10-22 三 12:30>                           :工作:
   do ...........
** TODO <2008-10-22 三 13:50>                           :工作:
   do ...........
** TODO <2008-10-22 三 20:43>                           :娱乐:
   do ...........
```

头脑碎片整理

```
#+STARTUP: overview
#+TAGS:  { 工作(w)  娱乐(p) 学习(s) }
#+TAGS:  { 当前(1) 近期(2) 远期(3) } { 容易(e) 麻烦(t) 困难(d) }
#+TAGS:
#+SEQ_TODO: 待办(T) 等待(W) | 完成(D) 取消(C) 委托(A)
#+COLUMNS: %10ITEM  %10PRIORITY %15TODO %65TAGS

* 待办 介绍Org                                          :工作:当前:
** 完成 说明                                             :容易:
   CLOSED: [2008-10-21 二 12:04]
** 完成 截图                                             :麻烦:
   CLOSED: [2008-10-21 二 12:04]
** 等待 举例                                             :困难:

* 学习                                              :学习:近期:
** 完成 DocBook                                             :麻烦:
   CLOSED: [2008-10-21 二 12:05]
** 取消 LaTex                                             :麻烦:
   CLOSED: [2008-10-21 二 12:05]

* 健身
** 等待 运动
   SCHEDULED: <2008-10-26 日>
** 取消 节食
   CLOSED: [2008-10-21 二 12:07]

* 编程
** 待办 C语言                                        :学习:近期:困难:
** 待办 Python                                        :学习:远期:麻烦:
** 取消 PHP                                        :学习:近期:麻烦:
   CLOSED: [2008-10-21 二 12:10]

* 临时
** org-mode                                        :工作:当前:麻烦:
   DEADLINE: <2008-10-21 二>
   要配上图片并举例
** 委托 Docbook介绍                                        :工作:当前:容易:
   CLOSED: [2008-10-21 二 12:12]
   有一点改动
```

使用 Org-mode 的时候其实不用这么一板一眼，可以随意一些，标签和日程表可以帮你把它们整理好

第 33 章 Zsh+screen

目录

screen简介

screen命令

screen操作

screen+zsh

在 `~/.zshrc` 文件中加入如下内容：

```
#screen integration to set caption bar dynamically
function title {
if [[ $TERM == "screen" || $TERM == "screen.linux" ]]; then
    # Use these two for GNU Screen:
    print -nR $'\033k'$1$'\033'\\\

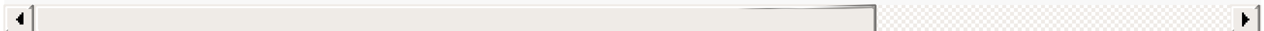
    print -nR $'\033]0;'$2$'\a'
elif [[ $TERM == "xterm" || $TERM == "urxvt" ]]; then
    # Use this one instead for XTerms:
    print -nR $'\033]0;'$*$'\a'
    #trap 'echo -ne "\e]0;$USER@$HOSTNAME: $BASH_COMMAND\007"' DEBUG
fi
}

#set screen title if not connected remotely
function precmd {
    title "`print -Pn "%~" | sed "s:\([~/][^/]*\)/.*/:\1...:"`" "$TERM $PWD"
    echo -ne '\033[?17;0;127c'
}

function preexec {
    emulate -L zsh
    local -a cmd; cmd=(${(z)1})
    if [[ $cmd[1]:t == "ssh" ]]; then
        title "@"$cmd[2] "$TERM $cmd"
    elif [[ $cmd[1]:t == "sudo" ]]; then
        title "#"$cmd[2]:t "$TERM $cmd[3,-1]"
    elif [[ $cmd[1]:t == "for" ]]; then
        title "()"$cmd[7] "$TERM $cmd"
    elif [[ $cmd[1]:t == "svn" ]]; then
        title "$cmd[1,2]" "$TERM $cmd"
    else
        title $cmd[1]:t "$TERM $cmd[2,-1]"
    fi
}
```

在 `~/.screenrc` 文件中加入如下内容：

```
hardstatus alwayslastline "%{=b}%{-b}%{R}[%{=b B}%-w%{=b BK}%>%n %t%{-}%+w%{-b}%< %=%{R}]
```



第 34 章 gentoo stage3

第 35 章 硬件问题

第 36 章 网络设置

第 37 章 自制 LiveCD

第 38 章 awesome

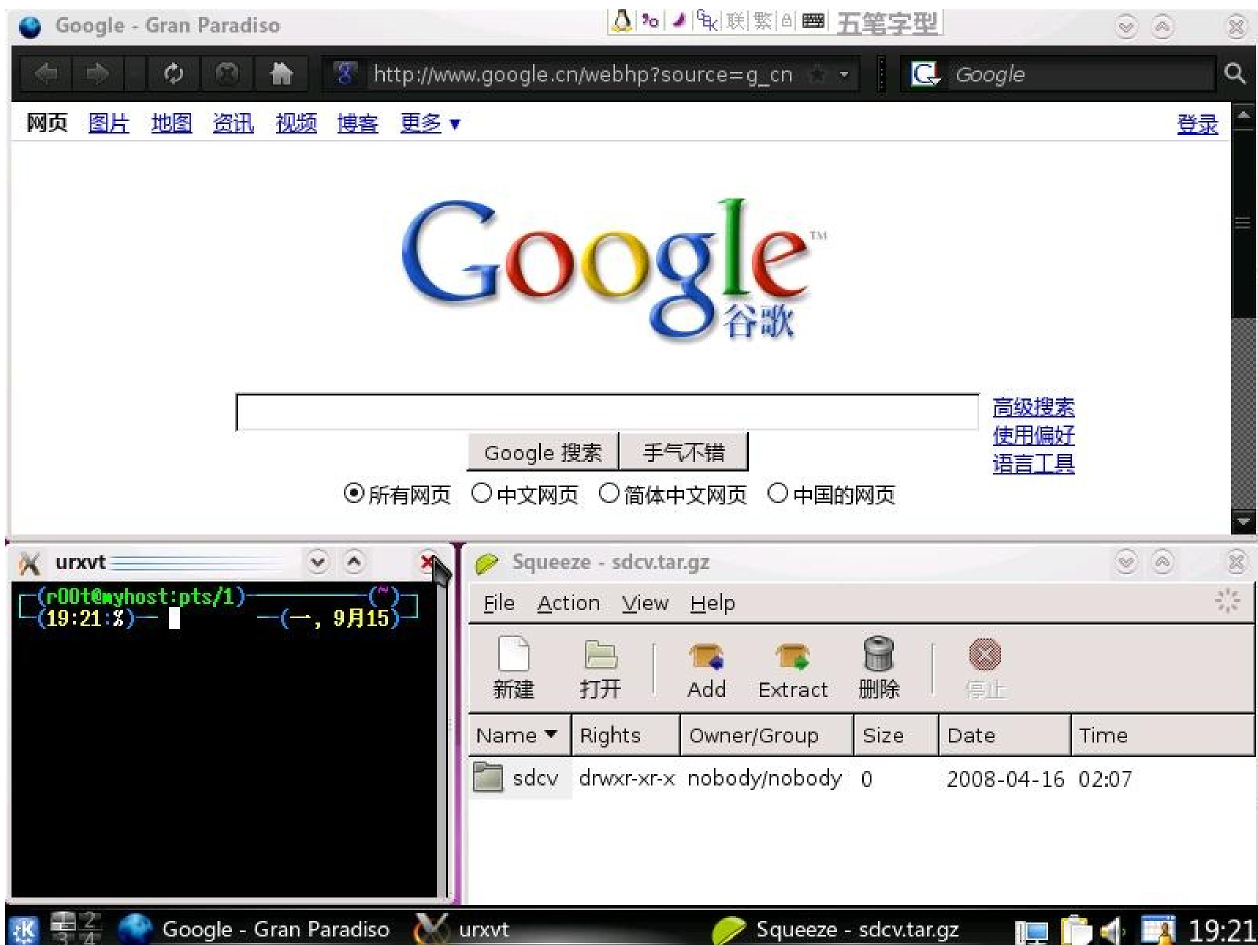
目录

Tiling window manager

`Tiling window manager` ，直译为“瓦片窗口管理器”[84]，意译为“平铺式窗口管理器”。

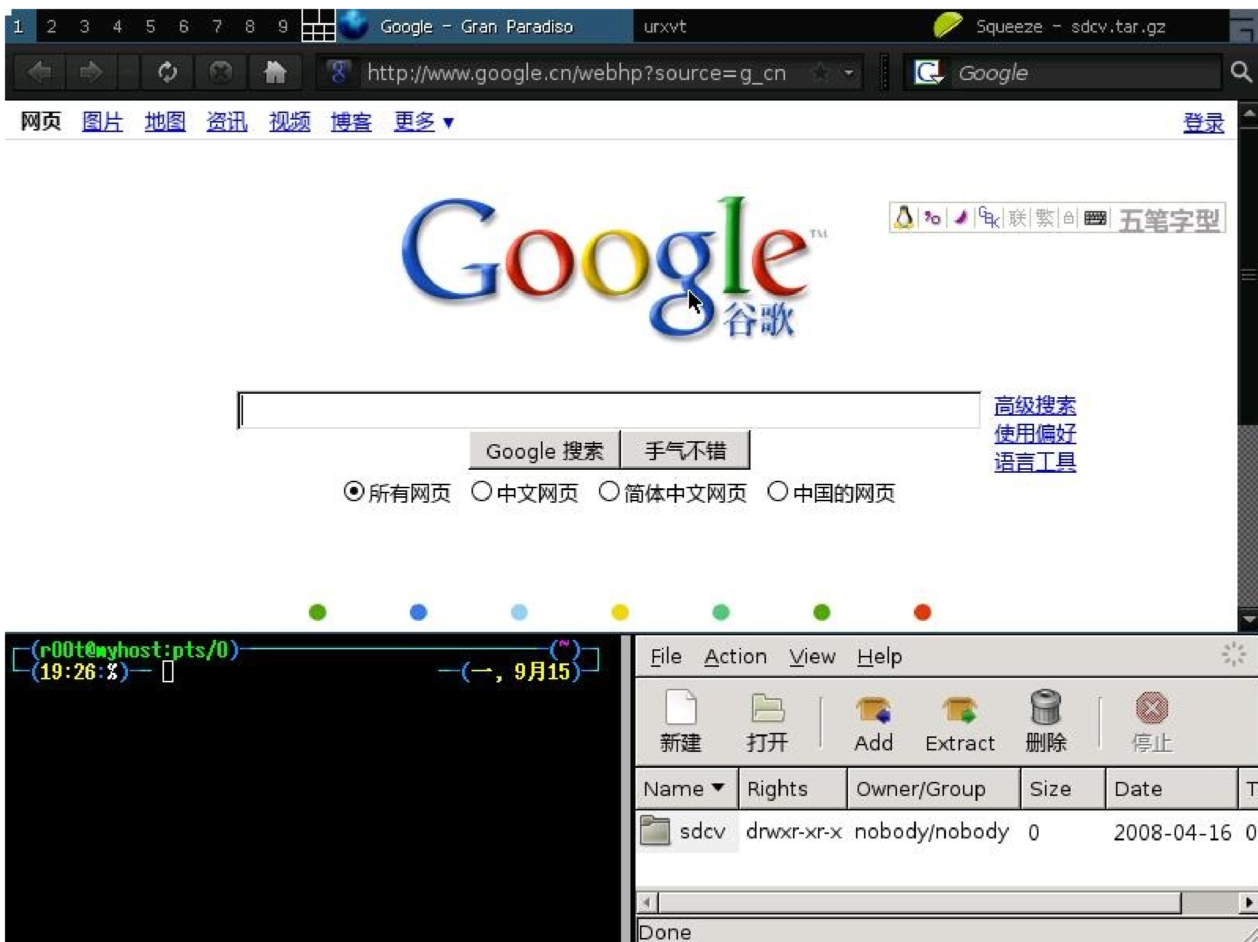
假设你需要同时监视多个程序的状态并进行操作，你就需要不停的按Alt+Tab切换，但是这会加速 Alt 尤其是 Tab 的磨损[85]。

无奈之下，你可能会这样作：



用鼠标拖放成这样，绝对不是一件愉快的事，而且各个窗口的大小可能不太合适，窗口之间可能会有点缝隙，虽然不漏风，但是也影响注意力......

而在平铺式窗口管理器中，无论打开多少个窗口，都会把整个屏幕占满，不留一点儿缝隙(如果只有一个窗口，直接全屏)



窗口没有标题栏（标题在状态栏上），边框尽可能的窄（可设置为0），最大程度的节约了桌面空间，如果你的显示器比较大，或者有多个显示器，平铺式窗口管理器将是不二之选

❸

常见的平铺式窗口管理器有：awesome、xmonad、dwm、wmii、ion、ratpoison、larswm等，推荐使用 `awesome`

> [84] 像屋顶的瓦片一样平铺开来
>
> [85] 手的磨损忽略了，因为它是可再生的

优点

awesome 有以下优点：

稳定、快速、简单

专注于效率

不需要为码放窗口分心，不需要在多个窗口间来回切换，不需要经常把手挪开使用鼠标

完全的键盘控制

在普通的窗口管理器中，完全键盘控制是不现实的，因为调整窗口位置无论如何要用鼠标。

强大、灵活的配置文件

使用 Lua 脚本作为配置文件使得它的灵活性非常高

支持多种窗口布局

多显示器支持

操作

awesome 中，所有的操作都可以用快捷键完成：

打开终端	Mod4 + Return	
运行命令	Mod4 + F1	
关闭当前窗口	Mod4 + Shift + c	
重启awesome	Mod4 + Control + r	
退出awesome	Mod4 + Shift + q	
重绘当前窗口	Mod4 + Shift + r	
窗口间切换	Mod4 + j	Mod4 + k
标签间切换	Mod4 + Left	Mod4 + Right
切换到标签	Mod4 + [1-9]	
屏幕间切换	Mod4 + Control + j	Mod4 + Control + k
切换布局	Mod4 + space	Mod4 + Shift + space
切换为浮动窗口	Mod4 + Control + space	
调整窗口位置	Mod4 + Shift + j	Mod4 + Shift + k
调整列大小	Mod4 + h	Mod4 + l
调整主区窗口数量	Mod4 + Shift + h	Mod4 + Shift + l
调整辅区窗口数量	Mod4 + Control + h	Mod4 + Control + l

配置

awesome 全局配置文件为 `/etc/xdg/awesome/rc.lua`，用户配置文件为 `~/.config/awesome/rc.lua`，以下是一些配置选项：

例 38.1. awesome 配置

```
-- 主题文件。awesome的主题非常简单，只需要指明几个颜色就可以了
theme_path = "/usr/local/share/awesome/themes/default"

-- 设置默认终端
terminal = "urxvt"

-- Mod4 对应“Win”键，可以改成其它的
modkey = "Mod4"

-- 不保留窗口间的缝隙
    c.honorsizehints = false

-- 默认使用浮动窗口的程序
-- 在awesome下用Mod4 + Ctr + i 查看当前程序的instance和class名称
floatapps =
{
    ["MPlayer"] = true,
    ["gimp"] = true,
    ["smplayer"] = true,
    ["mocp"] = true,
    ["Codeblocks"] = true,
-- 各种对话框
    ["Dialog"] = true,
-- firefox 的下载窗口
    ["Download"] = true,
    ["empathy"] = true
}

-- 程序启动时自动发送到某个屏幕的某个tag里
apptags =
{
    ["smplayer"] = { screen = 1, tag = 7 },
    ["amarokapp"] = { screen = 1, tag = 8 },
    ["VirtualBox"] = { screen = 1, tag = 9 },
    ["Firefox"] = { screen = 1, tag = 1},
    ["Thunderbird-bin"] = { screen = 1, tag = 7 },
    ["Linux-fetion"] = { screen = 1, tag = 6 },
}

-- 显示时间
-- 默认显示的是从1970年1月1日到现在经过了多少秒
-- 查找awful.hooks.timer.register这一行，改为如下配置
awful.hooks.timer.register(1, function ()
    mytextbox.text = " " .. os.date() .. " "
end)
```

第 39 章 openbox 工作环境

目录

为什么要用OpenBox:

- 速度非常快，资源占用极少。
- 可高度定制化，能够对应用程序加以灵活控制。
- 能够绑定键盘和鼠标。比如，你可以为程序设定启动快捷键，也可以为最小/大化窗口设置热键。对于鼠标，同样如此。
- 具有自动启动脚本，能够随机自动启动各种程序。

安装

```
pacman -S openbox openbox-themes obconf  feh pypanel
conky
```

*obconf*是OpenBox的图形化配置工具；*feh*用来设置桌面背景；*pypanel*是任务栏面板；*conky*用来监控系统状态。

OpenBox菜单

~/.config/openbox/menu.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<openbox_menu xmlns="http://openbox.org/3.4/menu">

<!--根菜单-->
<menu id="root-menu" label="Openbox Root Menu">
<!--分隔符 使用属性"label"显示标题 如果没有则显示为一条分隔线-->
  <separator label="Applications" />
<!--菜单中的条目-->
  <item label="Emacs">
    <action name="Execute">
      <command>emacs</command>
      <startupnotify>
```

```
          <enabled>yes</enabled>
          <wmclass>Emacs</wmclass>
        </startupnotify>
      </action>
    </item>
    <item label="Rxvt Unicode">
      <action name="Execute">
        <command>urxvt</command>
      </action>
    </item>
    <item label="Firefox">
      <action name="Execute">
        <command>firefox</command>
        <startupnotify>
          <enabled>yes</enabled>
          <wmclass>Firefox</wmclass>
        </startupnotify>
      </action>
    </item>
    <item label="MPlayer">
      <action name="Execute">
        <command>gmplayer</command>
        <startupnotify>
          <enabled>yes</enabled>
          <wmclass>MPlayer</wmclass>
        </startupnotify>
      </action>
    </item>
    <item label="Thunar">
      <action name="Execute">
        <command>Thunar</command>
        <startupnotify>
          <enabled>yes</enabled>
        </startupnotify>
      </action>
    </item>
    <item label="Squeeze">
      <action name="Execute">
        <command>squeeze</command>
        <startupnotify>
          <enabled>yes</enabled>
        </startupnotify>
      </action>
    </item>
    <item label="Evince">
      <action name="Execute">
        <command>evince</command>
        <startupnotify>
          <enabled>yes</enabled>
        </startupnotify>
      </action>
    </item>
    <item label="Gqview">
      <action name="Execute">
        <command>gqview</command>
        <startupnotify>
          <enabled>yes</enabled>
        </startupnotify>
      </action>
    </item>
    <item label="Sunbird">
      <action name="Execute">
        <command>sunbird</command>
        <startupnotify>
          <enabled>yes</enabled>
        </startupnotify>
      </action>
    </item>
      <separator />
  <!--子菜单 子菜单的内容在后面定义-->
    <menu id="alternate"/>
    <separator label="System" />
```

```
  <menu id="system-menu"/>
  <separator />
  <item label="Log Out">
    <action name="SessionLogout">
      <prompt>yes</prompt>
    </action>
  </item>
</menu>

<!--子菜单内容定义 与根菜单同级-->
<menu id="alternate" label="Alternate">
  <item label="GVim">
    <action name="Execute">
      <command>gvim</command>
      <startupnotify>
        <enabled>yes</enabled>
        <wmclass>GVim</wmclass>
      </startupnotify>
    </action>
  </item>
  <item label="Opera">
    <action name="Execute">
      <command>opera</command>
      <startupnotify>
        <enabled>yes</enabled>
        <wmclass>Opera</wmclass>
      </startupnotify>
    </action>
  </item>
  <item label="Rox">
    <action name="Execute">
      <command>rox</command>
      <startupnotify>
        <enabled>yes</enabled>
        <wmclass>ROX-Filer</wmclass>
      </startupnotify>
    </action>
  </item>
  <item label="Xterm">
    <action name="Execute"><command>xterm</command></action>
  </item>
</menu>

<menu id="system-menu" label="System">
  <item label="Reconfigure">
    <action name="Reconfigure" />
  </item>
    <item label="Openbox Configuration Manager">
    <action name="Execute">
      <command>obconf</command>
      <startupnotify><enabled>yes</enabled></startupnotify>
    </action>
  </item>
  <item label="Xfce Settings">
    <action name="Execute">
      <command>xfce-setting-show</command>
      <startupnotify><enabled>yes</enabled></startupnotify>
    </action>
  </item>
  <separator />
  <item label="Exit Openbox">
    <action name="Exit">
      <prompt>yes</prompt>
    </action>
  </item>
</menu>

</openbox_menu>
```

OpenBox自动运行

~/.config/openbox/autostart.sh

```
pypanel &
eval `feh --bg-scale wallpaper.jpg` &
conky &
```

OpenBox配置

~/.config/openbox/rc.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Do not edit this file, it will be overwritten on install.
        Copy the file to $HOME/.config/openbox/ instead. -->
<openbox_config xmlns="http://openbox.org/3.4/rc">
  <resistance>
    <strength>10</strength>
    <screen_edge_strength>20</screen_edge_strength>
  </resistance>
  <focus>
    <focusNew>yes</focusNew>
    <!-- always try to focus new windows when they appear. other rules do
       apply -->
    <followMouse>no</followMouse>
    <!-- move focus to a window when you move the mouse into it -->
    <focusLast>yes</focusLast>
    <!-- focus the last used window when changing desktops, instead of the one
       under the mouse pointer. when followMouse is enabled -->
    <underMouse>no</underMouse>
    <!-- move focus under the mouse, even when the mouse is not moving -->
    <focusDelay>200</focusDelay>
    <!-- when followMouse is enabled, the mouse must be inside the window for
       this many milliseconds (1000 = 1 sec) before moving focus to it -->
    <raiseOnFocus>no</raiseOnFocus>
    <!-- when followMouse is enabled, and a window is given focus by moving the
       mouse into it, also raise the window -->
  </focus>
  <placement>
    <policy>Smart</policy>
    <!-- 'Smart' or 'UnderMouse' -->
    <center>yes</center>
    <!-- whether to place windows in the center of the free area found or
       the top left corner -->
    <monitor>Any</monitor>
    <!-- with Smart placement on a multi-monitor system, try to place new windows
       on: 'Any' - any monitor, 'Mouse' - where the mouse is, 'Active' - where
       the active window is -->
  </placement>
  <theme>
    <name>Mikachu</name>
    <titleLayout>NLIMC</titleLayout>
    <!--
      available characters are NDSLIMC, each can occur at most once.
      N: window icon
      L: window label (AKA title).
      I: iconify
      M: maximize
      C: close
      S: shade (roll up/down)
      D: omnipresent (on all desktops).
  -->
    <keepBorder>yes</keepBorder>
```

```
    <animateIconify>yes</animateIconify>
    <font place="ActiveWindow">
      <name>sans</name>
      <size>8</size>
      <!-- font size in points -->
      <weight>bold</weight>
      <!-- 'bold' or 'normal' -->
      <slant>normal</slant>
      <!-- 'italic' or 'normal' -->
    </font>
    <font place="InactiveWindow">
      <name>sans</name>
      <size>8</size>
      <!-- font size in points -->
      <weight>bold</weight>
      <!-- 'bold' or 'normal' -->
      <slant>normal</slant>
      <!-- 'italic' or 'normal' -->
    </font>
    <font place="MenuHeader">
      <name>sans</name>
      <size>9</size>
      <!-- font size in points -->
      <weight>normal</weight>
      <!-- 'bold' or 'normal' -->
      <slant>normal</slant>
      <!-- 'italic' or 'normal' -->
    </font>
    <font place="MenuItem">
      <name>sans</name>
      <size>9</size>
      <!-- font size in points -->
      <weight>normal</weight>
      <!-- 'bold' or 'normal' -->
      <slant>normal</slant>
      <!-- 'italic' or 'normal' -->
    </font>
    <font place="OnScreenDisplay">
      <name>sans</name>
      <size>9</size>
      <!-- font size in points -->
      <weight>bold</weight>
      <!-- 'bold' or 'normal' -->
      <slant>normal</slant>
      <!-- 'italic' or 'normal' -->
    </font>
  </theme>
  <desktops>
    <!-- this stuff is only used at startup, pagers allow you to change them
       during a session

       these are default values to use when other ones are not already set
       by other applications, or saved in your session

       use obconf if you want to change these without having to log out
       and back in -->
    <number>4</number>
    <firstdesk>1</firstdesk>
    <names>
      <!-- set names up here if you want to, like this:
    <name>desktop 1</name>
    <name>desktop 2</name>
    -->
    </names>
    <popupTime>875</popupTime>
    <!-- The number of milliseconds to show the popup for when switching
       desktops.  Set this to 0 to disable the popup. -->
  </desktops>
  <resize>
    <drawContents>yes</drawContents>
    <popupShow>Nonpixel</popupShow>
    <!-- 'Always', 'Never', or 'Nonpixel' (xterms and such) -->
```

```
      <popupPosition>Center</popupPosition>
      <!-- 'Center', 'Top', or 'Fixed' -->
      <popupFixedPosition>
        <!-- these are used if popupPosition is set to 'Fixed' -->
        <x>10</x>
        <!-- positive number for distance from left edge, negative number for
           distance from right edge, or 'Center' -->
        <y>10</y>
        <!-- positive number for distance from top edge, negative number for
           distance from bottom edge, or 'Center' -->
      </popupFixedPosition>
    </resize>
    <!-- You can reserve a portion of your screen where windows will not cover when
       they are maximized, or when they are initially placed.
       Many programs reserve space automatically, but you can use this in other
       cases. -->
    <margins>
      <top>0</top>
      <bottom>0</bottom>
      <left>0</left>
      <right>0</right>
    </margins>
    <dock>
      <position>TopLeft</position>
      <!-- (Top|Bottom)(Left|Right|)|Top|Bottom|Left|Right|Floating -->
      <floatingX>0</floatingX>
      <floatingY>0</floatingY>
      <noStrut>no</noStrut>
      <stacking>Above</stacking>
      <!-- 'Above', 'Normal', or 'Below' -->
      <direction>Vertical</direction>
      <!-- 'Vertical' or 'Horizontal' -->
      <autoHide>no</autoHide>
      <hideDelay>300</hideDelay>
      <!-- in milliseconds (1000 = 1 second) -->
      <showDelay>300</showDelay>
      <!-- in milliseconds (1000 = 1 second) -->
      <moveButton>Middle</moveButton>
      <!-- 'Left', 'Middle', 'Right' -->
    </dock>
    <keyboard>
      <chainQuitKey>C-g</chainQuitKey>
      <!-- Keybindings for desktop switching -->
      <keybind key="C-A-Left">
        <action name="DesktopLeft">
          <dialog>no</dialog>
          <wrap>no</wrap>
        </action>
      </keybind>
      <keybind key="C-A-Right">
        <action name="DesktopRight">
          <dialog>no</dialog>
          <wrap>no</wrap>
        </action>
      </keybind>
      <keybind key="C-A-Up">
        <action name="DesktopUp">
          <dialog>no</dialog>
          <wrap>no</wrap>
        </action>
      </keybind>
      <keybind key="C-A-Down">
        <action name="DesktopDown">
          <dialog>no</dialog>
          <wrap>no</wrap>
        </action>
      </keybind>
      <keybind key="S-A-Left">
        <action name="SendToDesktopLeft">
          <dialog>no</dialog>
          <wrap>no</wrap>
        </action>
```

```
    </keybind>
    <keybind key="S-A-Right">
      <action name="SendToDesktopRight">
        <dialog>no</dialog>
        <wrap>no</wrap>
      </action>
    </keybind>
    <keybind key="S-A-Up">
      <action name="SendToDesktopUp">
        <dialog>no</dialog>
        <wrap>no</wrap>
      </action>
    </keybind>
    <keybind key="S-A-Down">
      <action name="SendToDesktopDown">
        <dialog>no</dialog>
        <wrap>no</wrap>
      </action>
    </keybind>
    <keybind key="W-F1">
      <action name="Desktop">
        <desktop>1</desktop>
      </action>
    </keybind>
    <keybind key="W-F2">
      <action name="Desktop">
        <desktop>2</desktop>
      </action>
    </keybind>
    <keybind key="W-F3">
      <action name="Desktop">
        <desktop>3</desktop>
      </action>
    </keybind>
    <keybind key="W-F4">
      <action name="Desktop">
        <desktop>4</desktop>
      </action>
    </keybind>
    <keybind key="W-d">
      <action name="ToggleShowDesktop"/>
    </keybind>
    <!-- Keybindings for windows -->
    <keybind key="A-F4">
      <action name="Close"/>
    </keybind>
    <keybind key="A-Escape">
      <action name="Lower"/>
      <action name="FocusToBottom"/>
      <action name="Unfocus"/>
    </keybind>
    <keybind key="A-space">
      <action name="ShowMenu">
        <menu>client-menu</menu>
      </action>
    </keybind>
    <!-- Keybindings for window switching -->
    <keybind key="A-Tab">
      <action name="NextWindow"/>
    </keybind>
    <keybind key="A-S-Tab">
      <action name="PreviousWindow"/>
    </keybind>
    <keybind key="C-A-Tab">
      <action name="NextWindow">
        <panels>yes</panels>
        <desktop>yes</desktop>
      </action>
    </keybind>
    <!-- Keybindings for running applications -->
    <keybind key="W-e">
      <action name="Execute">
```

```
        <startupnotify>
          <enabled>true</enabled>
          <name>Emacs</name>
        </startupnotify>
        <command>emacs</command>
      </action>
    </keybind>
    <keybind key="W-u">
      <action name="Execute">
        <command>urxvt</command>
      </action>
    </keybind>
    <keybind key="W-f">
      <action name="Execute">
        <command>firefox</command>
      </action>
    </keybind>
    <keybind key="W-s">
      <action name="Execute">
        <command>sunbird</command>
      </action>
    </keybind>
  </keyboard>
  <mouse>
    <dragThreshold>8</dragThreshold>
    <!-- number of pixels the mouse must move before a drag begins -->
    <doubleClickTime>200</doubleClickTime>
    <!-- in milliseconds (1000 = 1 second) -->
    <screenEdgeWarpTime>400</screenEdgeWarpTime>
    <!-- Time before changing desktops when the pointer touches the edge of the
       screen while moving a window, in milliseconds (1000 = 1 second).
       Set this to 0 to disable warping -->
    <context name="Frame">
      <mousebind button="A-Left" action="Press">
        <action name="Focus"/>
        <action name="Raise"/>
      </mousebind>
      <mousebind button="A-Left" action="Click">
        <action name="Unshade"/>
      </mousebind>
      <mousebind button="A-Left" action="Drag">
        <action name="Move"/>
      </mousebind>
      <mousebind button="A-Right" action="Press">
        <action name="Focus"/>
        <action name="Raise"/>
        <action name="Unshade"/>
      </mousebind>
      <mousebind button="A-Right" action="Drag">
        <action name="Resize"/>
      </mousebind>
      <mousebind button="A-Middle" action="Press">
        <action name="Lower"/>
        <action name="FocusToBottom"/>
        <action name="Unfocus"/>
      </mousebind>
      <mousebind button="A-Up" action="Click">
        <action name="DesktopPrevious"/>
      </mousebind>
      <mousebind button="A-Down" action="Click">
        <action name="DesktopNext"/>
      </mousebind>
      <mousebind button="C-A-Up" action="Click">
        <action name="DesktopPrevious"/>
      </mousebind>
      <mousebind button="C-A-Down" action="Click">
        <action name="DesktopNext"/>
      </mousebind>
      <mousebind button="A-S-Up" action="Click">
        <action name="SendToDesktopPrevious"/>
      </mousebind>
      <mousebind button="A-S-Down" action="Click">
```

```
        <action name="SendToDesktopNext"/>
      </mousebind>
    </context>
    <context name="Titlebar">
      <mousebind button="Left" action="Press">
        <action name="Focus"/>
        <action name="Raise"/>
      </mousebind>
      <mousebind button="Left" action="Drag">
        <action name="Move"/>
      </mousebind>
      <mousebind button="Left" action="DoubleClick">
        <action name="ToggleMaximizeFull"/>
      </mousebind>
      <mousebind button="Middle" action="Press">
        <action name="Lower"/>
        <action name="FocusToBottom"/>
        <action name="Unfocus"/>
      </mousebind>
      <mousebind button="Up" action="Click">
        <action name="Shade"/>
        <action name="FocusToBottom"/>
        <action name="Unfocus"/>
        <action name="Lower"/>
      </mousebind>
      <mousebind button="Down" action="Click">
        <action name="Unshade"/>
        <action name="Raise"/>
      </mousebind>
      <mousebind button="Right" action="Press">
        <action name="Focus"/>
        <action name="Raise"/>
        <action name="ShowMenu">
          <menu>client-menu</menu>
        </action>
      </mousebind>
    </context>
    <context name="Top">
      <mousebind button="Left" action="Press">
        <action name="Focus"/>
        <action name="Raise"/>
        <action name="Unshade"/>
      </mousebind>
      <mousebind button="Left" action="Drag">
        <action name="Resize">
          <edge>top</edge>
        </action>
      </mousebind>
    </context>
    <context name="Left">
      <mousebind button="Left" action="Press">
        <action name="Focus"/>
        <action name="Raise"/>
      </mousebind>
      <mousebind button="Left" action="Drag">
        <action name="Resize">
          <edge>left</edge>
        </action>
      </mousebind>
    </context>
    <context name="Right">
      <mousebind button="Left" action="Press">
        <action name="Focus"/>
        <action name="Raise"/>
      </mousebind>
      <mousebind button="Left" action="Drag">
        <action name="Resize">
          <edge>right</edge>
        </action>
      </mousebind>
    </context>
    <context name="Bottom">
```

```
        <mousebind button="Left" action="Press">
          <action name="Focus"/>
          <action name="Raise"/>
        </mousebind>
        <mousebind button="Left" action="Drag">
          <action name="Resize">
            <edge>bottom</edge>
          </action>
        </mousebind>
        <mousebind button="Middle" action="Press">
          <action name="Lower"/>
          <action name="FocusToBottom"/>
          <action name="Unfocus"/>
        </mousebind>
        <mousebind button="Right" action="Press">
          <action name="Focus"/>
          <action name="Raise"/>
          <action name="ShowMenu">
            <menu>client-menu</menu>
          </action>
        </mousebind>
      </context>
      <context name="BLCorner">
        <mousebind button="Left" action="Press">
          <action name="Focus"/>
          <action name="Raise"/>
        </mousebind>
        <mousebind button="Left" action="Drag">
          <action name="Resize"/>
        </mousebind>
      </context>
      <context name="BRCorner">
        <mousebind button="Left" action="Press">
          <action name="Focus"/>
          <action name="Raise"/>
        </mousebind>
        <mousebind button="Left" action="Drag">
          <action name="Resize"/>
        </mousebind>
      </context>
      <context name="TLCorner">
        <mousebind button="Left" action="Press">
          <action name="Focus"/>
          <action name="Raise"/>
          <action name="Unshade"/>
        </mousebind>
        <mousebind button="Left" action="Drag">
          <action name="Resize"/>
        </mousebind>
      </context>
      <context name="TRCorner">
        <mousebind button="Left" action="Press">
          <action name="Focus"/>
          <action name="Raise"/>
          <action name="Unshade"/>
        </mousebind>
        <mousebind button="Left" action="Drag">
          <action name="Resize"/>
        </mousebind>
      </context>
      <context name="Client">
        <mousebind button="Left" action="Press">
          <action name="Focus"/>
          <action name="Raise"/>
        </mousebind>
        <mousebind button="Middle" action="Press">
          <action name="Focus"/>
          <action name="Raise"/>
        </mousebind>
        <mousebind button="Right" action="Press">
          <action name="Focus"/>
          <action name="Raise"/>
```

```
      </mousebind>
    </context>
    <context name="Icon">
      <mousebind button="Left" action="Press">
        <action name="Focus"/>
        <action name="Raise"/>
        <action name="Unshade"/>
        <action name="ShowMenu">
          <menu>client-menu</menu>
        </action>
      </mousebind>
      <mousebind button="Right" action="Press">
        <action name="Focus"/>
        <action name="Raise"/>
        <action name="ShowMenu">
          <menu>client-menu</menu>
        </action>
      </mousebind>
    </context>
    <context name="AllDesktops">
      <mousebind button="Left" action="Press">
        <action name="Focus"/>
        <action name="Raise"/>
        <action name="Unshade"/>
      </mousebind>
      <mousebind button="Left" action="Click">
        <action name="ToggleOmnipresent"/>
      </mousebind>
    </context>
    <context name="Shade">
      <mousebind button="Left" action="Press">
        <action name="Focus"/>
        <action name="Raise"/>
      </mousebind>
      <mousebind button="Left" action="Click">
        <action name="ToggleShade"/>
      </mousebind>
    </context>
    <context name="Iconify">
      <mousebind button="Left" action="Press">
        <action name="Focus"/>
        <action name="Raise"/>
      </mousebind>
      <mousebind button="Left" action="Click">
        <action name="Iconify"/>
      </mousebind>
    </context>
    <context name="Maximize">
      <mousebind button="Left" action="Press">
        <action name="Focus"/>
        <action name="Raise"/>
        <action name="Unshade"/>
      </mousebind>
      <mousebind button="Middle" action="Press">
        <action name="Focus"/>
        <action name="Raise"/>
        <action name="Unshade"/>
      </mousebind>
      <mousebind button="Right" action="Press">
        <action name="Focus"/>
        <action name="Raise"/>
        <action name="Unshade"/>
      </mousebind>
      <mousebind button="Left" action="Click">
        <action name="ToggleMaximizeFull"/>
      </mousebind>
      <mousebind button="Middle" action="Click">
        <action name="ToggleMaximizeVert"/>
      </mousebind>
      <mousebind button="Right" action="Click">
        <action name="ToggleMaximizeHorz"/>
      </mousebind>
```

```
    </context>
    <context name="Close">
      <mousebind button="Left" action="Press">
        <action name="Focus"/>
        <action name="Raise"/>
        <action name="Unshade"/>
      </mousebind>
      <mousebind button="Left" action="Click">
        <action name="Close"/>
      </mousebind>
    </context>
    <context name="Desktop">
      <mousebind button="Up" action="Click">
        <action name="DesktopPrevious"/>
      </mousebind>
      <mousebind button="Down" action="Click">
        <action name="DesktopNext"/>
      </mousebind>
      <mousebind button="A-Up" action="Click">
        <action name="DesktopPrevious"/>
      </mousebind>
      <mousebind button="A-Down" action="Click">
        <action name="DesktopNext"/>
      </mousebind>
      <mousebind button="C-A-Up" action="Click">
        <action name="DesktopPrevious"/>
      </mousebind>
      <mousebind button="C-A-Down" action="Click">
        <action name="DesktopNext"/>
      </mousebind>
      <mousebind button="Left" action="Press">
        <action name="Focus"/>
        <action name="Raise"/>
      </mousebind>
      <mousebind button="Right" action="Press">
        <action name="Focus"/>
        <action name="Raise"/>
      </mousebind>
    </context>
    <context name="Root">
      <!-- Menus -->
      <mousebind button="Middle" action="Press">
        <action name="ShowMenu">
          <menu>client-list-combined-menu</menu>
        </action>
      </mousebind>
      <mousebind button="Right" action="Press">
        <action name="ShowMenu">
          <menu>root-menu</menu>
        </action>
      </mousebind>
    </context>
    <context name="MoveResize">
      <mousebind button="Up" action="Click">
        <action name="DesktopPrevious"/>
      </mousebind>
      <mousebind button="Down" action="Click">
        <action name="DesktopNext"/>
      </mousebind>
      <mousebind button="A-Up" action="Click">
        <action name="DesktopPrevious"/>
      </mousebind>
      <mousebind button="A-Down" action="Click">
        <action name="DesktopNext"/>
      </mousebind>
    </context>
  </mouse>
  <menu>
    <!-- You can specify more than one menu file in here and they are all loaded,
       just don't make menu ids clash or, well, it'll be kind of pointless -->
    <!-- default menu file (or custom one in $HOME/.config/openbox/) -->
    <file>menu.xml</file>
```

```
    <hideDelay>200</hideDelay>
    <!-- if a press-release lasts longer than this setting (in milliseconds), the
       menu is hidden again -->
    <middle>no</middle>
    <!-- center submenus vertically about the parent entry -->
    <submenuShowDelay>100</submenuShowDelay>
    <!-- this one is easy, time to delay before showing a submenu after hovering
       over the parent entry -->
    <applicationIcons>yes</applicationIcons>
    <!-- controls if icons appear in the client-list-(combined-)menu -->
    <manageDesktops>yes</manageDesktops>
    <!-- show the manage desktops section in the client-list-(combined-)menu -->
  </menu>
  <applications>
    <!--
  # this is an example with comments through out. use these to make your
  # own rules, but without the comments of course.

  <application name="first element of window's WM_CLASS property (see xprop)"
              class="second element of window's WM_CLASS property (see xprop)"
               role="the window's WM_WINDOW_ROLE property (see xprop)"
               type="the window's _NET_WM_WINDOW_TYPE (if unspecified, then
                       it is dialog for child windows)">
  # the name or the class can be set, or both. this is used to match
  # windows when they appear. role can optionally be set as well, to
  # further restrict your matches.

  # the name, class, and role use simple wildcard matching such as those
  # used by a shell. you can use * to match any characters and ? to match
  # any single character.

  # the type is one of: normal, dialog, splash, utility, menu, toolbar, dock,
  #    or desktop

  # when multiple rules match a window, they will all be applied, in the
  # order that they appear in this list

    # each element can be left out or set to 'default' to specify to not
    # change that attribute of the window

    <decor>yes</decor>
    # enable or disable window decorations

    <shade>no</shade>
    # make the window shaded when it appears, or not

    <position force="no">
      # the position is only used if both an x and y coordinate are provided
      # (and not set to 'default')
      # when force is "yes", then the window will be placed here even if it
      # says you want it placed elsewhere.  this is to override buggy
      # applications who refuse to behave
      <x>center</x>
      # a number like 50, or 'center' to center on screen. use a negative number
      # to start from the right (or bottom for <y>), ie -50 is 50 pixels from the
      # right edge (or bottom).
      <y>200</y>
      <monitor>1</monitor>
      # specifies the monitor in a xinerama setup.
      # 1 is the first head, or 'mouse' for wherever the mouse is
    </position>

    <focus>yes</focus>
    # if the window should try be given focus when it appears. if this is set
    # to yes it doesn't guarantee the window will be given focus. some
    # restrictions may apply, but Openbox will try to

    <desktop>1</desktop>
    # 1 is the first desktop, 'all' for all desktops

    <layer>normal</layer>
    # 'above', 'normal', or 'below'
```

```
    <iconic>no</iconic>
    # make the window iconified when it appears, or not

    <skip_pager>no</skip_pager>
    # asks to not be shown in pagers

    <skip_taskbar>no</skip_taskbar>
    # asks to not be shown in taskbars. window cycling actions will also
    # skip past such windows

    <fullscreen>yes</fullscreen>
    # make the window in fullscreen mode when it appears

    <maximized>true</maximized>
    # 'Horizontal', 'Vertical' or boolean (yes/no)
  </application>

  # end of the example
-->
  </applications>
</openbox_config>
```

conky配置

~/.config/openbox/.conkyrc

```
# set to yes if you want Conky to be forked in the background
background yes

cpu_avg_samples 2
net_avg_samples 2

out_to_console no

# X font when Xft is disabled, you can pick one with program xfontsel
#font 7x12
#font 6x10
#font 7x13
#font 8x13
#font 7x12
#font *mintsmild.se*
#font -*-*-*-*-*-*-34-*-*-*-*-*-*-*
#font -artwiz-snap-normal-r-normal-*-*-100-*-*-p-*-iso8859-1

# Use Xft?
use_xft yes

# Xft font when Xft is enabled
#xftfont Bitstream Vera Sans Mono:size=8
xftfont Monospace:size=8

own_window_transparent yes
own_window_colour hotpink
# Text alpha when using Xft
xftalpha 0.8

#on_bottom yes

# mail spool
mail_spool $MAIL

# Update interval in seconds
update_interval 1
# Create own window instead of using desktop (required in nautilus)
own_window yes
```

```
#If own_window is yes, you may use type normal, desktop or override
#own_window_type normal
#own_window_type desktop
own_window_type override

# Use double buffering (reduces flicker, may not work for everyone)
double_buffer yes

# Minimum size of text area
#minimum_size 280 5
maximum_width 300

# Draw shades?
draw_shades no

# Draw outlines?
draw_outline no

# Draw borders around text
draw_borders no

# Stippled borders?
stippled_borders 10

# border margins
border_margin 4

# border width
border_width 1

# Default colors and also border colors
default_color white
default_shade_color white
default_outline_color white

# Text alignment, other possible values are commented
#alignment top_left
#minimum_size 10 10
gap_x 10
gap_y 10
alignment top_right
#alignment bottom_left
#alignment bottom_right

# Gap between borders of screen and text

# Add spaces to keep things from moving about?  This only affects certain objects.
use_spacer no

# Subtract file system buffers from used memory?
no_buffers yes

# set to yes if you want all text to be in uppercase
uppercase no

# boinc (seti) dir
# seti_dir /opt/seti

# Possible variables to be used:
#
#      Variable         Arguments                  Description
#  acpiacadapter                        ACPI ac adapter state.
#  acpifan                              ACPI fan state
#  acpitemp                             ACPI temperature.
#  adt746xcpu                           CPU temperature from therm_adt746x
#  adt746xfan                           Fan speed from therm_adt746x
#  battery           (num)              Remaining capasity in ACPI or APM
#                                       battery. ACPI battery number can be
#                                       given as argument (default is BAT0).
#  buffers                              Amount of memory buffered
#  cached                               Amount of memory cached
#  color             (color)            Change drawing color to color
```

```
#  cpu                                  CPU usage in percents
#  cpubar            (height)           Bar that shows CPU usage, height is
#                                       bar's height in pixels
#  downspeed         net                Download speed in kilobytes
#  downspeedf        net                Download speed in kilobytes with one
#                                       decimal
#  exec              shell command      Executes a shell command and displays
#                                       the output in torsmo. warning: this
#                                       takes a lot more resources than other
#                                       variables. I'd recommend coding wanted
#                                       behaviour in C and posting a patch :-).
#  execi             interval, shell    Same as exec but with specific interval.
#                    command            Interval can't be less than
#                                       update_interval in configuration.
#  fs_bar            (height), (fs)     Bar that shows how much space is used on
#                                       a file system. height is the height in
#                                       pixels. fs is any file on that file
#                                       system.
#  fs_free           (fs)               Free space on a file system available
#                                       for users.
#  fs_free_perc      (fs)               Free percentage of space on a file
#                                       system available for users.
#  fs_size           (fs)               File system size
#  fs_used           (fs)               File system used space
#  hr                (height)           Horizontal line, height is the height in
#                                       pixels
#  i2c               (dev), type, n     I2C sensor from sysfs (Linux 2.6). dev
#                                       may be omitted if you have only one I2C
#                                       device. type is either in (or vol)
#                                       meaning voltage, fan meaning fan or temp
#                                       meaning temperature. n is number of the
#                                       sensor. See /sys/bus/i2c/devices/ on
#                                       your local computer.
#  kernel                               Kernel version
#  loadavg           (1), (2), (3)      System load average, 1 is for past 1
#                                       minute, 2 for past 5 minutes and 3 for
#                                       past 15 minutes.
#  machine                              Machine, i686 for example
#  mails                                Mail count in mail spool. You can use
#                                       program like fetchmail to get mails from
#                                       some server using your favourite
#                                       protocol. See also new_mails.
#  mem                                  Amount of memory in use
#  membar            (height)           Bar that shows amount of memory in use
#  memmax                               Total amount of memory
#  memperc                              Percentage of memory in use
#  new_mails                            Unread mail count in mail spool.
#  nodename                             Hostname
#  outlinecolor      (color)            Change outline color
#  pre_exec          shell command      Executes a shell command one time before
#                                       torsmo displays anything and puts output
#                                       as text.
#  processes                            Total processes (sleeping and running)
#  running_processes                    Running processes (not sleeping),
#                                       requires Linux 2.6
#  shadecolor        (color)            Change shading color
#  stippled_hr       (space),           Stippled (dashed) horizontal line
#                    (height)
#  swapbar           (height)           Bar that shows amount of swap in use
#  swap                                 Amount of swap in use
#  swapmax                              Total amount of swap
#  swapperc                             Percentage of swap in use
#  sysname                              System name, Linux for example
#  time              (format)           Local time, see man strftime to get more
#                                       information about format
#  totaldown         net                Total download, overflows at 4 GB on
#                                       Linux with 32-bit arch and there doesn't
#                                       seem to be a way to know how many times
#                                       it has already done that before torsmo
#                                       has started.
#  totalup           net                Total upload, this one too, may overflow
#  updates                              Number of updates (for debugging)
```

```
#  upspeed          net             Upload speed in kilobytes
#  upspeedf         net             Upload speed in kilobytes with one
#                                   decimal
#  uptime                           Uptime
#  uptime_short                     Uptime in a shorter format
#
#  seti_prog                        Seti@home current progress
#  seti_progbar     (height)        Seti@home current progress bar
#  seti_credit                      Seti@hoome total user credit

# variable is given either in format $variable or in ${variable}. Latter
# allows characters right after the variable and must be used in network
# stuff because of an argument
#${font Dungeon:style=Bold:pixelsize=10}I can change the font as well
#${font Verdana:size=10}as many times as I choose
#${font Perry:size=10}Including UTF-8,
#${font Luxi Mono:size=10}justo como este texto que o google traduz fêz o português
# stuff after 'TEXT' will be formatted on screen
#${font Grunge:size=12}${time %a  %b  %d}${alignr -25}${time %k:%M}

TEXT
${color lightgrey}${font Monospace:style=Bold:size=8}${time %b%d日 星期%a}${alignr}${time
${color #ddaa00}${exec whoami}${color snow} on ${color #88aadd}$nodename${alignr}$sysname
${color #88aadd}$stippled_hr
${color lightgrey}Uptime:$color $uptime ${color lightgrey}- Load:$color $loadavg
${color lightgrey}Usage:${cpu cpu1}%  ${cpu cpu2}%  ${acpitemp}°C${offset 50}
${color yellow}${cpugraph cpu1 25,150 32cd32 228b22}${alignr}${cpugraph cpu2 25,150 32cd3
${color lightgrey}RAM Usage:$memperc% $mem/$memmax${alignr}Disk IO: ${diskio}/s
${color #b8860b}${memgraph 25,150 4682b4 4682b4}${alignr}${color #cd950c}${diskiograph 25
${color lightgrey}Swap Usage:    $swapperc% ${alignr}$swap/$swapmax
${color #ffe7ba}${swapbar}
${color lightgrey}File System:    /     ${alignr}${fs_used /}/${fs_size /}
${color lightgrey}${fs_bar /}
${color lightgrey}File System:    /home ${alignr}${fs_used /home}/${fs_size /home}
${color lightgrey}${fs_bar /home}
${color lightgrey}Processes:$color $processes  ${color grey}Running:$color $running_proce
${color}CPU usage            PID    CPU%   MEM%
${color #ddaa00} ${top name 1} ${top pid 1} ${top cpu 1} ${top mem 1}
${color lightgrey} ${top name 2} ${top pid 2} ${top cpu 2} ${top mem 2}
${color lightgrey} ${top name 3} ${top pid 3} ${top cpu 3} ${top mem 3}
${color}Mem usage
${color #ddaa00} ${top_mem name 1} ${top_mem pid 1} ${top_mem cpu 1} ${top_mem mem 1}
${color lightgrey} ${top_mem name 2} ${top_mem pid 2} ${top_mem cpu 2} ${top_mem mem 2}
${color lightgrey} ${top_mem name 3} ${top_mem pid 3} ${top_mem cpu 3} ${top_mem mem 3}
${color #88aadd}$stippled_hr
${color lightgrey}Networking:$alignr${color snow}${addr ppp0}
${color lightgrey}Down:${color #8844ee} ${downspeed eth0} k/s${color lightgrey} ${offset
${color darkgreen}${downspeedgraph eth0 25,150 ff0000 0000ff} $alignr${color darkgreen}${
${color #1e90ff}Totaldown:${color #09ff09}${totaldown eth0} ${color #1e90ff}${offset 20}T
${color #88aadd}$stippled_hr
${rss http://feed.feedsky.com/my_cnbeta 5 item_titles 10}
${color #ddaa00}Port(s): Connections:${color #5d478b}  Inbound: ${tcp_portmon 1 32767 cou
${color #5d478b}Inbound Connection | Local Service/Port | Outbound Connection | Remote Se
${color darkgreen}${tcp_portmon 1 32767 rhost 0} ${tcp_portmon 1 32767 lservice 0} ${tcp_
${tcp_portmon 1 32767 rhost 1} ${tcp_portmon 1 32767 lservice 1} ${tcp_portmon 32768 6100
${tcp_portmon 1 32767 rhost 2} ${tcp_portmon 1 32767 lservice 2} ${tcp_portmon 32768 6100
${tcp_portmon 1 32767 rhost 3} ${tcp_portmon 1 32767 lservice 3} ${tcp_portmon 32768 6100
${tcp_portmon 1 32767 rhost 4} ${tcp_portmon 1 32767 lservice 4} ${tcp_portmon 32768 6100
${tcp_portmon 1 32767 rhost 5} ${tcp_portmon 1 32767 lservice 5} ${tcp_portmon 32768 6100
```

第 40 章 Emacs muse

第 41 章 写作工具链

第 42 章 使用 lftp

目录

lftp 简介

lftp是个功能强大的字符界面文档传输工具，它包含以下功能：

- 支持ftp、ftps、http、https、hftp、fish等传输协议
- 支持FXP
- 支持代理
- 支持多线程传输
- 支持书签
- 类似bash，提供后台命令、nohop模式、命令历史、命令别名、命令补齐等进程管理功能

登录 ftp服务器

使用以下命令登录 ftp服务器：

```
lftp ftp://用户名[:密码]@服务器地址[:端口] #标准方式，推荐
lftp 用户名[:密码]@服务器地址[:端口]
lftp 服务器地址 [-p 端口] -u 用户名[,密码]
lftp 服务器地址[:端口] -u 用户名[,密码]
```

- 如果不指定端口，默认 `21`
- 如果不在命令中使用明文输入密码，连接时会询问密码(推荐)

可以使用“书签”收藏服务器站点，在 lftp 中以下命令，为当前站点定义别名：

```
lftp >bookmark             #显示所有收藏
lftp >bookmark add 别名   #使用 `别名` 收藏当前站点
```

使用别名登录 ftp服务器：

```
lftp 别名
```

也可以编辑 lftp 的配置文件 `~/.lftp/bookmarks` ，格式如下:

```
别名 ftp://用户名:密码@服务器地址:端口
```

lftp 使用方法

大多数图形界面的 ftp客户端，都有两栏窗口，一栏为本地目录，一栏为远程目录。lftp 也采用这种方式工作，只不过没有使用图形界面直观的显示

命令	本地	远程
显示工作目录	lpwd	pwd
切换目录	lcd	cd
显示文件列表	!ls	cls
!ls -l	ls	

- 其中， `!` 表示执行本地命令，lftp中没有与 **ls** 对应的本地命令 **lls**， 所以要使用 **!ls** 显示本地目录文件

使用以上命令确认当前工作目录的情况。以下命令用于从本地目录上传，或者从远程目录下载：

	下载	上传
单个文件	get	put
多个文件	mget	mput
多线程	pget	
目录	mirror	mirror -R

- 在 lftp 配置文件 `~/.lftp/rc` 中设置 **pget** 使用的线程数

  ```
  set pget:default-n 5
  ```

在远程目录中，可以使用以下命令操作文件

统计文件大小	du
移动、重命名	mv
删除	rm
创建文件夹	mkdir
删除文件夹	rmdir

使用 `exit` 命令退出 lftp

中文乱码

大多数 windows 平台下的 ftp服务器 使用 GB2312 编码，而 lftp 使用 UTF-8 编码，使用 lftp 访问这些服务器，中文显示为乱码。可以通过指定编码来解决

```
lftp >set ftp:charset gbk   #设置远程编码为gbk
lftp >set file:charset utf8 #设置本地编码(Linux系统默认使用 UTF-8，这一步通常可以省略)
```

也可以在 lftp 配置文件中 `~/.lftp/rc` 设置默认值：

```
set ftp:charset gbk
set file:charset utf8
```

第 43 章 Firefox 使用技巧

目录

Firefox高级设置

urlclassifier3.sqlite

urlclassifier3.sqlite文件用于记录Firefox从Google抓取的反钓鱼网站和恶意站点数据的,但是这个文件大小在默认情况下会不断地增长,通过设置"urlclassifier.updatecachemax"可以限制urlclassifier3.sqlite的大小.

在Linux版本下"urlclassifier.updatecachemax"默认为104857600 (100 MB)

在地址栏输入about:config，出现一个警告，点击同意后进行高级设置。在过滤器中输入上面字串，改它的值为20971520（20m）。删除urlclassifier3.sqlite重新启动Firefox就可以了

扩展

我用的扩展

名称	说明
Fission	在地址栏显示载入进度条（Safari风格）
Hide Menubar	自动隐藏菜单栏
NASA Night Launch	主题，制作相当精良
Vimperator	让Firefox模仿Vim的工作方式

其实不必要安装太多的扩展，那样只会让Firefox越来越臃肿、迟钝。我所选用的扩展带有明确的目的性，那就是尽量腾出更多的空间。

第 44 章 FVWM

部分 IV. 地质

目录

第 45 章 Unix

目录

作者:不详

1965年时，贝尔实验室(Bell Labs)加入一项由奇异电子(General Electric)和麻省理工学院(MIT)合作的计画；该计画要建立一套多使用者、多任务、多层次(multi-user、multi-processor、multi-level)的MULTICS操作系统。直到1969年，因MULTICS计画的工作进度太慢，该计画就被停了下来。当时，Ken Thompson（后被称为Unix之父）已经有一个称为「星际旅行」的程序在GE-635的机器上跑，但是反应非常的慢，正巧也被他发现了一部被闲置的PDP-7(Digital的主机)，Ken Thompson和Dernis Ritchie就将「星际旅行」的程序移植到PDP-7上。而这部PDP-7就此在整个计算机历史上留下了芳名。

MULTICS 其实是"MULTiplexed Information and Computing System"的缩写，在1970年时，那部PDP-7却只能支持两个使用者，当时，Brian Kernighan 就开玩笑地戏称他们的系统其实是："UNiplexed Information and Computing System"，缩写为"UNICS"，后来，大家取其谐音，就称其为"Unix"了。1970年可称为是Unix元年。

1971 年，他们申请了一部PDP-11/20，申请的名义是：要发展文书处理系统。该提案被获采纳，他们也发展出了一套文书处理系统 — 就是现在Unix操作系统里面文书处理系统(nroff/troff)的前身。有趣的是，没有多久，贝尔实验室的专利部门真的采用了这套系统作为他们处理文件的工具，而贝尔实验室的专利部门也就顺理成章地成为Unix的第一个正式使用者。当时，那部PDP-11／20只有0.5MB磁盘空间。而描述这整个系统的文件被标示为："First Edition"，版本日期是1970年11月。从此以后，Unix的版本就以系统文件的版别来称呼。

发展

Unix操作系统的历史漫长而曲折，它的第一个版本是1969年由Ken Thompson在AT&T贝尔实验室实现的，运行在一台DEC PDP-7计算机上。这个系统非常粗糙，与现代Unix相差很远，它只具有操作系统最基本的一些特性。后来Ken Thompson和Dennis Ritchie使用C语言对整个系统进行了再加工和编写，使得Unix能够很容易的移植到其他硬件的计算机上。从那以后，Unix系统开始了令人瞩目的发展。

由于此时 AT&T还没有把Unix作为它的正式商品，因此研究人员只是在实验室内部使用并完善它。正是由于Unix是被作为研究项目，其他科研机构和大学的计算机研究人员也希望能得到这个系统，以便进行自己的研究。AT&T以分发许可证的方法，对Unix仅仅收取很少的费用，大学和研究机构就能获得Unix的源代码以进行研究。Unix的源代码被散发到各个大学，一方面使得科研人员能够根据需要改进系统，或者将其移植到其他的硬件环境中去，另一方面培养了懂得Unix使用和编程的大量的学生，这使得Unix的普及更为广泛。

由于操作系统的开发相当困难，只有少数的计算机厂商，如 IBM、Digital等大型公司，才拥有自己的操作系统，而其他众多生产计算机的硬件厂商则采用别人开发的操作系统。因为Unix不需要太多的花费，因此很多厂商就选择了Unix作为他们生产的计算机使用的操作系统。他们把Unix移植到自己的硬件环境下，而不必从头开发一个操作系统。

到了 70年代末，在Unix发展到了版本6之后，AT&T认识到了Unix的价值，成立了Unix系统实验室（Unix System Lab,USL）来继续发展Unix。因此AT&T一方面继续发展内部使用的Unix版本7，一方面由USL开发对外正式发行的Unix版本，同时AT&T也宣布对Unix产品拥有所有权。几乎在同时，加州大学伯克利分校计算机系统研究小组（CSRG）使用Unix对操作系统进行研究，因此他们的研究成果就反映在他们使用的Unix中。他们对Unix的改进相当多，增加了很多当时非常先进的特性，包括更好的内存管理，快速且健壮的文件系统等，大部分原有的源代码都被重新写过，以支持这些新特性。很多其他Unix使用者，包括其他大学和商业机构，都希望能得到CSRG改进的Unix系统。因此CSRG中的研究人员把他们的Unix组成一个完整的Unix系统——BSD Unix（Berkeley Software Distribution），向外发行。

BSD Unix在Unix的历史发展中具有相当大的影响力，被很多商业厂家采用，成为很多商用Unix的基础,而AT&T与其同时存在的Unix版本的影响就小得多。同时很多研究项目也是以BSD Unix为研究系统，例如美国国防部的项目—ARPANET，ARPANET今天发展成为了Internet，而BSD Unix中最先实现了TCP/IP，使Internet和Unix紧密结合在一起。

而 AT&T的Unix系统实验室，同时也在不断改进他们的商用Unix版本，直到他们吸收了BSD Unix中已有的各种先进特性，并结合其本身的特点，推出了Unix System V版本之后，情况才有了改变。从此以后，BSD Unix和Unix System V形成了当今Unix的两大主流，现代的Unix版本大部分都是这两个版本的衍生产品。

Unix的版本号表示方式比较复杂，各种不同的Unix版本有自己的版本标识方式，如最早AT&T开发的内部Unix使用简单的顺序号来标识版本，从V 1到V 7。

BSD使用主版本加次版本的方法标识，如4.2BSD，4.3BSD，在原始版本的基础上还有派生版本，这些版本通常有自己的名字，如4.3BSD-Net/1，4.3BSD-Net/2。

AT&T使用罗马数字来标识他们的对外发布的Unix版本，用Release来表示次版本。如System V Release 4（简写为SVR4）标识AT&T的Unix System V的第四次发布的版本。

其他商业公司的 Unix使用自己的版本标识，如Sun的Solaris 2.6，IBM的AIX 4.0等。

虽然 AT&T的Unix System V也是非常优秀的Unix版本，但是BSD Unix在Unix领域内的影响更大。AT&T的Unix系统实验室一直关注着BSD的发展，在1992年，Unix系统实验室指控BSDI── 一家发行商业BSD Unix的公司，违反了AT&T的许可权，发布自己的Unix版本，并进一步指控伯克利计算机系统研究组泄漏了Unix的商业机密（此时的 4.3BSD中来自AT&T Unix的代码已经不足10%）。这个官司影响了很多Unix厂商，使他们不得不从BSD Unix转向Unix System V,以避免法律问题。以至于当今大多数商业Unix版本都是基于Unix System V的。

这件有关 Unix版权的案子直到Unix系统实验室被AT&T卖给了Novell公司后才得以解决，Novell不打算陷入这样的法律纷争中，因此就采用了比较友好的做法。伯克利的CSRG被允许自由发布BSD，但是其中来自于AT&T的代码必须完全删除。因此CSRG就对他们最新的4.4BSD 进行了修改，删除了那些来自于AT&T的源代码，发布了4.4 BSD Lite版本（该系统是不完整的，尤其对于英特尔386体系的计算机系统）。由于这个版本不存在法律问题，4.4BSD Lite成为了现代BSD系统的基础版本。

Novell的比较友善的做法还不止对BSD，他把自己的Unix改名为Unixware，而将Unix商标赠送给X/Open── 一个由众多Unix厂家组成的联盟，这样这个联盟内的所有成员均可使用Unix商标。从此之后，Unix不再是专有产品的名字了。同时，由于BSD系统已经十分成熟，作为对操作系统进行研究的目标已经达到，伯克利计算机系统研究组（CSRG）在发布了4.4BSD-lite2之后就解散了，小组的科研人员有些进入了Unix商业公司，有些继续进行其他计算机领域的研究。此时，严格意义上的Unix System V和BSD Unix都不复存在了，存在的只是他们的各种后续版本。

回顾 Unix的发展，可以注意到Unix与其他商业操作系统的不同之处主要在于其开放性。在系统开始设计时就考虑了各种不同使用者的需要，因而Unix被设计为具备很大可扩展性的系统。由于它的源码被分发给大学，从而在教育界和学术界影响很大，进而影响到商业领域中。大学生和研究者为了科研目的或个人兴趣在 Unix上进行各种开发，并且不计较金钱利益，将这些源码公开，互相共享，这些行为极大丰富了Unix本身。很多计算机领域的科学家和技术人员遵循这些方式，开发了数以千计的自由软件，包括FreeBSD在内。正因为如此，当今的Internet才如此丰富多采，与其他商业网络不同，才能成为真正的全球网络。开放是Unix的灵魂，也是Internet的灵魂。

由于 Unix的开放性，另一方面就使得存在多个不同的Unix版本。由于不同的Unix使用稍有差别的文件、目录结构，提供略有不同的系统调用，因此对系统管理、以及为Unix开发可移植的应用程序带来一定的困难。例如System V和BSD的很多系统调用就存在不同。在Unix历史发展中也存在将Unix完全统一的努力，例如POSIX规范就是各个Unix厂商经过协商，达成的Unix操作系统应该遵循的一套基本系统调用的规定。然而由于存在多个Unix系统，各个厂商的意见很不统一，因此POSIX规范制定的很宽松，甚至 Windows NT中也存在一个POSIX子系统。事实上，只要各个Unix之间协调发展，不故意为了商业目的而人为的制造系统差异，就能够保持各个Unix之间不致具有太大的差别，保持Unix系统的多样性比只有唯一的一个Unix系统更能够促进技术的进步和发展，企图人为的统一Unix的想法只能是某一个Unix厂商的一厢情愿。

有时为了避免法律纠纷，表明自己的程序代码是完全重写的，一个软件厂商会故意将一些函数声明的与其他系统的中的同类函数不同，例如使用不同的函数参数或改变函数参数的顺序等。 Unix System V中的很多系统调用与BSD Unix不同的部分原因也在于此。这些差异人为造成了不同系统的源程序的差异。

UNIX家谱

UNIX的历史开始于1969年ken Thompson，Dennis Ritchie（即著名的K&G，C语言的发明人）与一群人在一部PDP-7上进行的一些工作，后来这个系统变成了UNIX。它主要的几个版本为：

V1（1971）：第一版的UNIX，以PDP-11/20的汇编语言写成。包括文件系统，fork、roff、ed等软件。

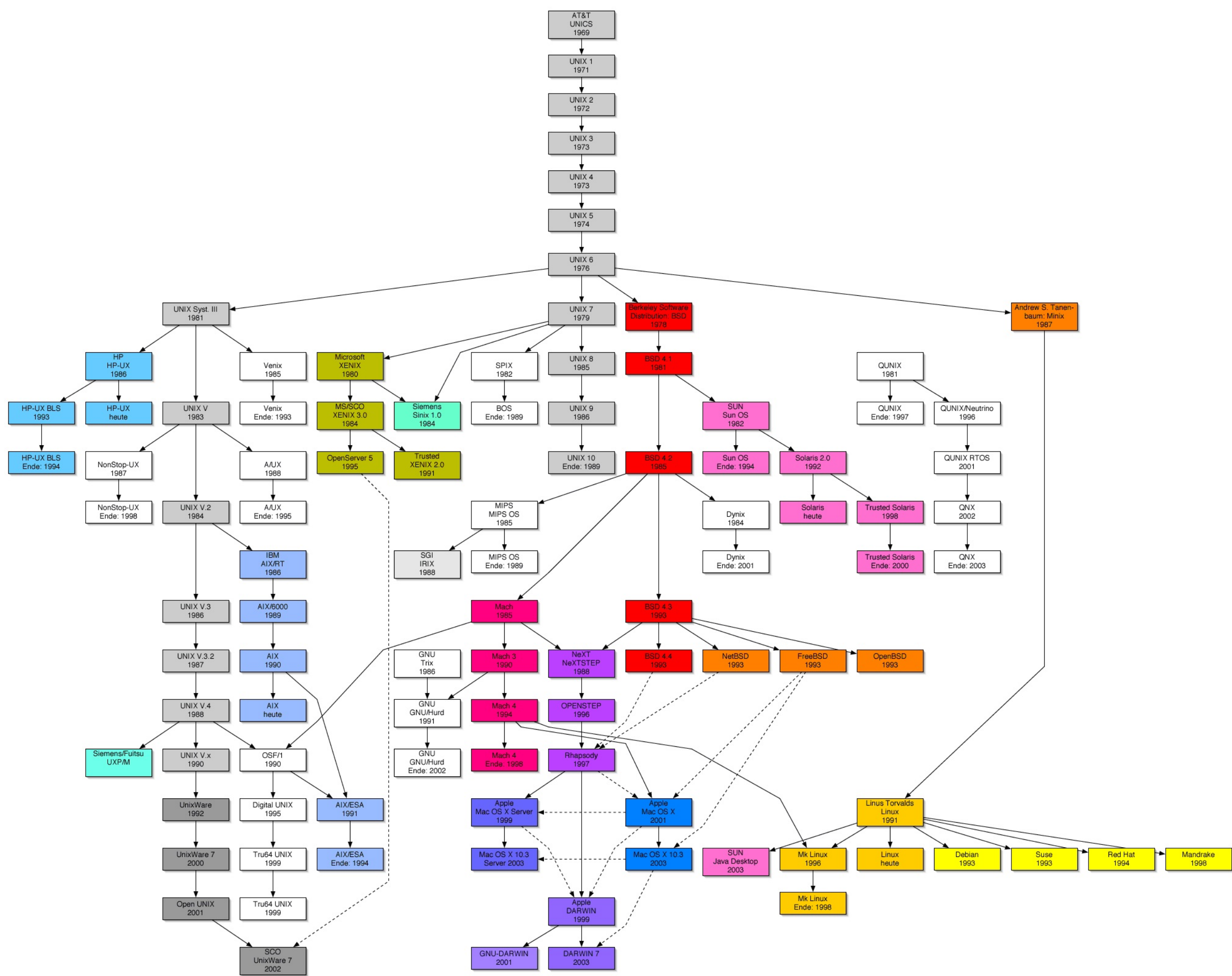
V4（1973）：以C语言从头写过，这使得UNIX修改容易，可以在几个月内移植到新的硬件平台上。最初C语言是为UNIX设计的，所以C与UNIX间有紧密的关系。

V6（1975）：第一个在贝尔实验室外（尤其是大学中）广为流传的UNIX版本。这也是UNIX分支的起点与广受欢迎的开始。1.xBSD（PDP-II）就是由这个版本衍生出来的。

V7（1979）：在许多UNIX玩家的心目中，这是“最后一个真正的UNIX，”这个版本包括一个完整的K&R C编译器，Bourne shell。V7移植到VAX机器后称为32V。

目前开发UNIX（System V）的公司是Unix System Laboratories (USL)。USL本为AT&T所有，1993年初被Novell收购。Novell于1993年末将UNIX这个注册商标转让给X/Open组织。

目前为止，UNIX有两大流派：那就是AT&T发布的UNIX操作系统System V与美国加州大学伯克利分校发布的UNIX版BSD（Berkeley Software Distribution）。SVR4是两大流派融合后的产物。1991年底，与System V针锋相对的开放软件基金会(Open Software Foundation)推出了OSF/1。



现在几种主要的UNIX版本

- AIX：IBM的UNIX，是根据SVR2（最近已经出到SVR3.2）以及一部分BSD延伸而来，加上各种硬件的支持。具备特有的系统管理（SMIT）。
- 386BSD：Jolitz从Net/2 software移植过来的。支持Posix，32位。
- FreeBSD：1.x从386BSD 0.1而来，FreeBSD 2.x版是用4.4BSD lite改写。
- HP-UX（HP）：旧系统是从S III（SVRx）发展面来，现在是由SVR2（4.2BSD）发展而来，目前是10.x版。
- Linux(x86)：遵从POSIX，SYSV及BSD的扩展
- OSF/1（DEC）：DEC对OSF/1的移植。
- SCO UNIX（x86）：SVR3.2，目前影响较大的PC UNIX。
- SunOS（680x0，Sparc，i386）：根据4.3BSD，包含许多来自System V的东西。Sun的主要成果在于：NFS，OpenLook GUI标准，现演变为Solaris 。
- Ultrix(DEC)：根据4.2BSD再加上许多4.3BSD的东西。
- Xenix(x86)：Intel硬件平台上的UNIX，以SVR2为基础，由微软推出。在中国使用较广泛。

附录: Unix 时代的开创者 Ken Thompson

自图灵奖诞生以来，其获得者一直都是计算机领域的科学家与学者，而在所有这些届的图灵奖中只有唯一的一届有个例外，那就是Ken Thompson与Dennis M. Ritchie，他们都是计算机软件工程师。

Ken Thompson，1943年出生于美国新奥尔良。1960年，Ken进入加州大学伯克利分校主修电气工程。1965年从伯克利毕业后，又花了一年的时间在该校取得了电子工程硕士的学位。不知道是时代造就英雄，还是英雄顺应时代而生，在Ken读书期间，正好赶上了计算机时代蓬勃发展的起步阶段，自小喜欢电气的Ken接触到计算机后，立即完全沉迷了进去，从1962年的开始，他就在学校的计算机中心找到份工作，专门负责程序的编写。这也为其后他一手开创的 Unix时代奠定了良好基础。

1966年离开校园的Ken加入了贝尔实验室。那时的计算机系统还是批处理的天下，程序员只能在又慢又笨重大型机上工作，一般来讲是先将程序卡片装入设备，然后再等1个小时再过来取回运算的结果，其效率之低可想而知。应市场的需要，当时贝尔实验室与麻省理工学院以及通用电气公司联合开发了一个多用户分时操作系统，取名为Multics(多路信息计算系统)，Ken当时就是这个系统的开发人员之一，在开发Multics的期间，Ken创造出了名为 Bon的编程语言。可惜因为这个系统不但开发周期长，成本高，而且庞大而缓慢，市场前景完全不被看好，最后贝尔实验室从这个项目中撤了出来。这对于Ken 而言，简直是个巨大的不幸，因为他自己用写的一个“star travel”游戏就是完全基于Multics的，退出Multics项目意味着Ken将没有机器可以再玩这个游戏了。

面对此情此景，Ken作为一个创造者的本性立即体现了出来，于是他决定自己写一个操作系统来满足他玩游戏的需要，说干就干，Ken找到了一台废弃已久的老式PDP-7，并在这台机器上重写了他的游戏。在这个过程中，Ken有了一个主意，要开发一个全新的操作系统。利用PDP-7上的汇编语言，Ken 只花了一个月就编写完了操作系统的内核，在这个一个月中，他一周一个内核，一个文件系统，一个编辑器和一个编译程序的完成。做完这个系统后，Ken将其命名为UNiplexed Information and Computing System，缩写为 UNICS，后来做了一下改动，称为UNIX，在开发第一版Unix的过程中，Ken还开发出一种新的语言，即C语言的前身——B语言，这种语言简洁明了，接近于硬件语言，第一版的Unix就是基于B语言来开发的。

Unix的出现开始虽然并不为大家所看好，但是却引起了贝尔实验室另一位同事的注意，这就是Dennis M. Ritchie，于是Dennis主动加入了进来共同完善这个系统。至此一场轰轰烈烈的Unix的传奇时代才真正的拉开了序幕。1972年，他们联手将 Unix移植到当时最先进的大型机PDP-2上，由于Unix是如此的简洁、稳定与高效，以至于当时大家都放弃了PDP-2上自带的DEC操作系统，而完全改用Unix，这时的Unix已经开始走向成熟了。在1973年之前Unix还不太为外界所知，到同年10月，Unix在IBM举办的操作系统原理专题研讨会上被提及，当Ken和Dennis在会上宣读论文并展示Unix后，整个会场轰动了，大家都立即涌上来索取这种新型的操作系统的程序。随着Unix 的需求量的日益增加，Ken与Dennis决定将Unix进一步改写，以便可以移植到各种不同的硬件系统，由于Unix的原码中不少是用汇编完成，不具备良好的移植性，正好Dennis在1973年在B语言的基础上开发出了C语言，C语言灵活，高效性，与硬件

无关，并且不失其简洁性，正是Unix移植所需要的法宝，于是旧版的Unix与C语言完美结合在一起产生了新的可移植的Unix系统。随着Unix的广泛使用，C语言也成为了当时最受欢迎的编程语言一直到延续至今。

说到Unix与C语言，还有一段小故事，当时安装了Unix的PDP-11被放在贝尔实验室供大家使用，有一天大家伙发现Ken总是可以得到最高的权限轻松进入他们的帐户，在贝尔实验室这种高人云集的地方，这简单是太不能容忍了，于是有若干高人跳了出来，仔细分析Unix代码，找到后门，修改后再重新编译整个Unix，当所有人都以为这个世界应该从此清静了的时候，却发现Ken还是很容易就取得了他们的帐户权限，为此大家都闷不已。至到很多年后，Ken才道出其中的原委，原来代码里确实存在后门，不过并不在Unix代码中，而是藏在编译Unix的编译器里，每次编译器编译时就会自动加入后门代码，而当时整个贝尔实验室都用的是Ken所写的C编译器。

由于Unix与C语言的深远影响，1983年美国计算机协会将当年的图灵奖破例颁给了作为软件工程师的Ken与Dennis，并在当年还决定新设立一个奖项——软件系统奖，以奖励那些优秀的软件开发者，当然首个软件系统奖也是非他们两人莫属了。

虽然Unix与C语言让Ken与Dennis功成名就，但是他们两人都没有走那些IT史上自己创业的通用套路，而是一直留在贝尔实验室从事其喜爱的软件开发工作。到了2000年12月时，Ken正式退休，离开了工作了几十年的贝尔实验室开始享受他晚年的时光，但是Ken怎么能闲得下来呢，于是他干脆将他的另一个爱好：飞机，变成正式的职业，成为了一名专职的飞行员。至此，他所开创的Unix时代已经完全与他无关了。

第 46 章 Gnu

第 47 章 软件业自由之神——Richard Stallman

目录

作者:方兴东

在 Richard Stallman 的理论下，用户彼此拷贝软件不但不是“盗版”，而是体现了人类天性的互助美德。对 Richard Stallman 来说，自由是根本，用户可自由共享软件成果，随便拷贝和修改代码。他说:“想想看，如果有人同你说:‘只要你保证不拷贝给其他人用的话，我就把这些宝贝拷贝给你。’其实，这样的人才是魔鬼；而诱人当魔鬼的，则是卖高价软件的人。”可以断定，进入世纪末，软件业发生的最大变革就是自由软件的全面复兴。在自由软件的浪潮下，软件业的商业模式将脱胎换骨，从卖程序代码为中心，转化为以服务为中心。

五短身材，不修边幅，过肩长发，连鬓胡子，时髦的半袖沙滩上装，一副披头士的打扮。看起来象现代都市里的野人。如果他将一件“麻布僧袍”穿在身上，又戴上一顶圆形宽边帽子，有如绘画作品中环绕圣像头上的光环。一眨眼的功夫，他又变成圣经中的耶稣基督的样子，散发着先知般的威严和力量。野人与基督，恰恰就是自由软件的精神领袖 Richard Stallman 的双重属性：他既是当今专有（私有）商业软件领域野蛮的颠覆者，又是无数程序员和用户心目中神圣的自由之神。

在他的理论下，用户彼此拷贝软件不但不是“盗版”，而是体现了人类天性的互助美德。对 Richard Stallman 来说，自由是根本，用户可自由共享软件成果，随便拷贝和修改代码。他说:“想想看，如果有人同你说:‘只要你保证不拷贝给其他人用的话，我就把这些宝贝拷贝给你。’其实，这样的人才是魔鬼；而诱人当魔鬼的，则是卖高价软件的人。”可以断定，进入世纪末，软件业发生的最大变革就是自由软件的全面复兴。在自由软件的浪潮下，软件业的商业模式将脱胎换骨，从卖程序代码为中心，转化为以服务为中心。

有人说， Richard Stallman 应该算是世界上最伟大，软件写得最多的程序设计师。但是，Richard Stallman 真正的力量，还是他的思想。

自由软件是计算机业的传统

自由软件不是新生事物，而是计算机业与身俱来的传统。纵览计算机发展史，从1946年到60年代，从 IBM 蓝色巨人到 ARPANET，从集成电路到 PC 机，从互联网到电信自由经营，每一个时期都留下了“自由”的影子。

可以说自由拷贝和源代码开放是整个计算机业，包括个人电脑及互联网两大领域的天然的软件开发和传播模式。早在60年代，就有以大学为阵地，以年轻人为主题，自由地交流的风尚，并在软件开发与研究方面硕果累累：如 Unix、TCP/IP、Fortran、Pascal、LISP 等等。

当70年代，AT&T 被迫退出计算机业时，KenThompson 和 DennisRitche 从贝尔实验室将 Unix 的源代码拿出来，结果吸引了成千上万名程序员，为其改进、修正、添加，诞生了多年来高端系统最核心的操作系统--Unix的繁荣。

70年代中期，个人电脑革命还在酝酿之中，当时的软件是鼓励自由拷贝的（那时还没有发明盗版的名目），正是这种自由拷贝、信息共享的精神上点燃了个人电脑革命，促成了软件业的发展。甚至连盖茨起家的 Basic 也是依靠这种自由软件才流行起来，才为事实标准。其实，后来的许多软件都是依靠共享方式才取得成功。

在互联网发展初期，程序员也是将源代码自由共享。当时基于 UUCP 的 UUCPNet 和基于 TCP/IP 的 ARPANet（互联网前身），都有 Usenet 社区，其目的就是共享源代码交流经验。进入90年代，奠定互联网爆炸的一些关键技术，如伯纳斯-李发明的 WWW 技术，浏览器以及 Apache、BIND 等等全部都不是诞生在专有软件世界里。可以说，是自由软件的精神和创新奠定了整个计算机业的核心。忽视自由软件的传统和作用是不科学的。

70年代末，微软公司的创始人比尔·盖茨《致电脑业余爱好者的一封公开信》为标志，以世界知识产权组织《伯尔尼公约》为框架，软件步入了 Copyright 的时代。随着现代商业软件的发展，对利润的疯狂追逐不但割裂了传统，极大地偏离了计算机的基本精神。而且还在不断变本加厉。因此，自由软件的复兴首先是对现有版权体系的强力反叛。

软件源代码是交流技术、交流思想的主要媒介，正象传统的科学是通过论文、著作进行交流一样。企业为了保护软件的知识产权而将源码秘而不宣，已经背离了知识产权保护创新的基本精神。

Richard Stallman 在《为什么软件不应有‘所有者’？》一文中指出，软件的编写者将软件“据为己有”看上去天经地义。但必须看到，一个软件并不是单纯的工具，一旦软件的编写者将其传播出去，就不简单地是在提供“工具”，而是在传播“思想”。在这一点上，现有的版权体系采用了所谓保护“表达（Expression）”，不保护“思想（Idea）”的两分法，为软件保护问题设置了无法解释的障碍，造成了软件的精神分裂。

同时，自由和共享也是计算机发展的内在精神和永恒的追求目标。有人指出：在互联网被标榜为“资源共享”、“资源优化”的利器的时代，却不能对软件产品真正实现“共享”和“优化”，这不能不说是一个巨大的讽刺。

黑客传统

StevenLevy的名著《黑客：电脑时代的英雄》论述了个人计算机兴起的历史。该书最后一章讲述了 Richard Stallman 的故事。题目就是："最后一个眞正黑客"。这是对他最恰当的评价。连他的反对者也说："如果不存在 Richard Stallman ，人类也应该把他创造出来。"

1971年，年轻、聪明绝顶的程序员 Richard Stallman 进入 MIT 人工智能实验室工作，成为软件共享社区的重要成员。其实这个社区已存在多年。当然，软件共享也只不过局限于这个特定的社区中。它与计算机的历史一样悠久。就象配料共享与烹饪一样古老。计算机业的传统就是：一切为人人所共享。私有让人嘲笑，专用受人鄙视。

当时，人工智能实验室使用一个 ITS （不兼容分时系统）分时操作系统。黑客们（不是大众媒体所谈的安全破坏分子，而是指酷爱编程的人）是用汇编语言为 Digital 的 PDP-10 设计和编写的。PDP-10 是当时最著名的计算机之一。作为社区成员和实验室的系统黑客，Richard Stallman 的工作就是改进系统。

当时没有人称它为自由软件，因为这个词还不存在。但实际上就是这么回事。无论是某个公司成员或另一所大学想获得它，大家都会非常高兴地把源程序给他。如果你看到别人使用一种你没见过且有意思的程序，你可以坦然地向他索要程序，这样你就可以读它、改它，或拆卸部分用于新的程序。进入80年代，这种自由发生急剧变化， DEC 的 PDP-10 系列发生中断了。它的自由体系架构，在60年代显得强劲、先进。但到80年代就捉襟见肘，没有足够多的地址空间。这意味着几乎所有的为 ITS 编写的程序都作废了。黑客社区也崩溃了。

然而，到80年代后，计算机的商业化和软件专有化席卷整个产业，黑客们的黄金时代结束了。一个又一个有才能的 MIT 编程员离开了校园，投入了市场的怀抱。尤其是 Symbolics 公司的成立，挖走了社区中的许多黑客，大大伤了 MIT 人工智能实验室的元气。Richard Stallman 感到：一个时代结束了。

Richard Stallman 说:"那时，人工智能实验室已没法再支持下去，我是最后一个还想让它活起来的呆瓜。但后来我也没办法了，因为一个人根本发挥不了作用。"开始，他觉得这样与现实抗争没有什么意义。但他终于看出，他眞正的敌人不是 Symbolics，而是整个不开放源代码的商业软件业。

1981年，当人工智能实验室购买了新的 PDP-10 时，领导决定用 Digital 专有的分时系统来代替ITS。当时的计算机，无论是 VAX 或 68020 都有他们自己的操作系统。但没有一个是自由软件：你要获得一份可执行的拷贝必须签署一份不准向外公开的协议。

这就意味着使用计算机就得承诺：不能帮助你的邻居和朋友。这是软软件业迈出的可怕的第一步。一个相互协作、彼此交流的社区就这样被禁止了。由专有软件所有者所制订的规则："如果你与你的邻居共享，你就是盗版者。如果你想作点改动，那你得乞求我们来做。"

究竟谁违背了道德

实际上，恰恰是专有软件的理念--不允许共享或改动软件--是反社会的，也是不道德的，而且也是完全错误的。但是长期以来，软件出版商使人们相信：软件天生就该如此。这种片面的认识禁锢了人们的思维。当他们在谈论如何加强版权或打击盗版时，他们也认定这是天经地义，人们也会毫无异议地接受。

他们的第一个假设就是：软件公司对自己的软件拥有毫无疑问的天然权力，因而可以将权利施加到所有用户身上。（因为如果是天然权力，那不管对公众会造成多大的损害，我们也不能加以反对。）但有意思的是，美国宪法和美国法律惯例否定了这种看法，版权不是一项天然权力，只是一项人为由政府施加的独占，他限制了用户拷贝的天然权力。

另一个潜在的假设是，软件唯一重要的事就是它允许你可以做什么。而我们的计算机用户不必考虑我们处在的社会状况，被动接受就行。第三个假设就是如果我们不允许软件公司给用户施加权力，我们就没有可用的软件。这个假设看起来似是而非。实际上当自由软件兴起后，我们无须戴上锁链就能获得大量优秀软件。

如果我们拒绝接受上述假设，并从“用户第一”的基本道德常识上来考虑问题，那么我们将得出截然不同的结论。计算机用户应该有自由根据自己的需求修正程序；用户有自由共享软件，因为帮助别人是社会的基础。而软件厂商不可以对用户施加压力，剥夺用户的各种自由。

Richard Stallman 经历过沉痛的遭遇。70年代，激光打印机大得像吉普车一样，所以 Xerox（施乐）送了一台图象激光打印机给人工智能实验室时，人们发现唯有人工智能实验室的九楼机房里，才找得到位置放它。在大楼里所有人只要在各自的电脑上打些指令，叫打印机帮你服务。

它的打印速度令人满意，只是有时纸印光了或夹了纸，一大堆人的列印工作就全部停了下来。有时有的人要印上一大堆东西，而有些人只要印一两张时，不得不爬上九楼，把印表机的控制改一下，使它先印一两张。于是一天就这样爬上爬下，没有人受得了。

幸好印表机送来时， Xerox 把驱动程序的源代码也随机附上，实验室的人就把控制打印机驱动程序的功能作了些修改，大家都省了不少麻烦和汗水。Richard Stallman 回忆地说:“你的打印工作做完后，它还会通知你；如果夹了纸或你想问些什么打印上的事情，它也会让你知道。”

1978年，一切都变了。 Xerox 送了一台叫 Dover 的新打印机机给人工智能实验室，但不愿再附上源代码。Richard Stallman 说:“因此我们没办法修改驱动程序，于是整个大楼的打印效率又回到从前，卡了纸或把纸印光了，你在下面也无法知道。”

他和人工智能实验室为给打印机添加功能以便机器更好工作，希望获得打印机控制程序的源代码，结果被严正拒绝。“因此，我再无法说服自己不公开协议是纯洁清白的。当他们拒绝与我们共享时，我十分气愤。我不能更更弦改辙，对别人做出同样的事。”

赤裸裸的道德抉择

“随着社区的终结，我面临着一个道德上的抉择。最简单的就是投身于专有软件世界之中，签署不公开协议，并承诺不帮助同行、同事。而且自己也很可能编写软件，并在不公开协议的前提下发布软件，去同流合污，迫使更多的人背叛自己的原则。显然，走这条路，可以挣大钱，而且使编写代码的工作增添一份金钱上的快乐。但是我知道，等到自己职业生涯终结时，我再回首这些年为分离人类而砌造的‘墙壁’。我会感受到，我将自己的一生都用在使这个世界变得更加糟糕。”

另一个选择，很直截了当，但令人不愉快，那就是从此离开计算机领域。“这样我的技能不会被滥用，但也将被浪费，我不会因为分化和限制计算机用户而感到有罪，但这些事情会继续发生。”“因此，我开始寻找一条出路，使程序员可以做真正的好事。我问自己，我能写什么软件，我能否让社区重焕生机。”

答案很明白：首先需要的是一个操作系统，这是开始使用计算机的关键软件。有了操作系统，就能做许多事，没有操作系统，计算机都无法运行。有了自由操作系统，我们就能再次组建一个相互合作的黑客社区。而且任何人使用自由软件都不必剥夺他/她与朋友家人的共享权利。

作为一名操作系统的开发人员， Richard Stallman 无疑最胜任。“虽然我没有认为自己一定能成功，但我意识到自己就是命定做这项工作的。”Richard Stallman 选择做一个与 Unix 兼容的操作系统。这样容易被移植，而且 Unix 用户可以方便地转移过来。 GNU 这个名字确定就是遵循黑客传统，是一个递归的缩略词：“GNU IS NOT Unix。”

一个操作系统并不仅仅意味着一个内核，而且仅能运行其他程序也是不够的。一个完整的操作系统，要有指令处理器、汇编程序、编译器、解释程序、调试器、文本编辑器、邮件软件等等一个完整的系统。

自由软件“freesoftware”是一个被广泛误解的词，这个“free”完全与价格无关，它指“自由”。这就象“自由讲演(freeSpeech)”与“免费啤酒(freeBeer)”的区别。其主要内涵就是用户可以自由运行软件，可以按自己的要求自由修改软件，用户也可以自己销售软件，不管是收费的还是免费的。自由软件与出售软件拷贝并不冲突。

开发一个完整的系统是项庞大的工程。Richard Stallman 决定尽可能采用已有的自由软件，比如一开始他将 Tex 作为主要的文本格式标识符，几年后他又用 XWindows 系统作为 GUN 的图口系统。

思想比代码更闪光，但没有代码，思想是没有躯体的。

一个人的战争

1984年1月，Richard Stallman 辞去了 MIT 的工作，他担心 MIT 会要求产品的所有权，会给产品强加入自己的销售条件，最终又会成为专有软件。一开始， GNU 计划只有他一个人。他发现自己原来在人工智能实验室的办公室，还没有分给其他人用时，他就每天晚上溜进去工作。久而久之，白天他也跑去用实验室里的电脑。

当时人工智能实验室主任 PatrickH. Winston 并不干涉。因为 Winston 始终不把 Richard Stallman 的辞职当真，只要 Richard Stallman 能创造些好东西给大家用，实在没有必要把这位共事13年的老同事打发走路。因此他爽快地邀请 Richard Stallman 可以继续使用实验室的设备。从此，Richard Stallman 就成了特殊的一员。

工程启动后，Richard Stallman 听到有一个自由大学编译器套件（VUCK）。他去信询问能否用入 GNU 。答复是嘲弄式的，说对大学是自由的，但软件本身不行。于是，决定他为 GUN 编写的第一个软件就是一个多语言、多平台的编译器。他想利用 Pastel 编译器的源代码，但最终放弃。从头编写了新的编译器，名为 GCC 。

1984年9月，Richard Stallman 开始 GUNEmacs，1985年初，它开始可以工作。这使它可以用 Unix 系统进行编辑。此时，人们开始想使用 Emacs 。因此一个现实的问题是：如何传播它？当然，他将其放到了 MIT 计算机的匿名服务器上。但那时互联网还未普及，人们很难通过 FTP 获得拷贝。而失业的 Richard Stallman 也需要收入。于是，他宣布任何人都可以用150美元的价格获得程序。自由软件的分销商业模式就此诞生。如今，整个基于 Linux 的 GUN 系统都是如此。

为防止不肖厂商利用自由软件，使其专有化。Richard Stallman 别出心裁，创造了 Copyleft 的授权办法。所有的 GNU 程序遵循一种“Copyleft”原则，即可以拷贝，可以修改，可以出售，只是有一条：源代码所有的改进和修改必须向每个用户公开，所有用户都可以获得改动后的源码。它保证了自由软件传播的延续性。

市场里出政权

EMACS这样的程序最难的是开头。一旦第一版本推出之后，就有一大堆人去玩它，然后精益求精，越改越好。目前已有几百种 EMACS 的副程序，可用在50多种电脑上，从微电脑到 Cray 的超级电脑都可用 EMACS 。

由于 EMACS 的成功，Richard Stallman 设立了个新的基金会:自由软件基金会（FreeSoftwareFoundation(FSF)）；捐助 FSF 和 GNU 计划的厂商，也可享有减税的优待。单单1989年， FSF 就收到267782美元的捐助，基金会也因出售 GNU 程序手册和电脑磁带，而赚了330377美元。此外，Richard Stallman 也不再天天溜回人工智能实验室“借”用电脑，因为许多厂家已为 FSF 提供一大堆的高性能工作站等硬件设备，包括 HP、ThinkingMachine、Sony，甚至 UNIX 的娘家---贝尔实验室，也贡献了不少设备。也有一些厂商捐赠现金，并把技术人员送到 FSF 来向 Richard Stallman 学习，而且支付 Richard Stallman 的员工薪水。

FSF 就用这些钱来养起14位基金会成员:9位程序设计师，3位负责技术资料撰写。虽然 Richard Stallman 自己不支薪，但他不能期望他的同仁也和他一样看得开，而饿着肚子为理想拼斗。 FSF 的程序设计师一年也只有2万5千美元的薪水，这是一般厂商的一半或三分之一。Richard Stallman 之所以以低薪待人，原因就是可多请几位志同道合的黑客，为理想而工作。

GNU 在工作站和微机市场很风光，许多 工作站UNIX 和微机厂家，都把 GNU 纳入他们操作系统，包括 ConvexComputer 、DEC、DataGeneral 及以前的 NeXT 等。

GNU 工程激励了许许多多年轻的黑客，他们编写了大量自由软件。最后，是里奴斯·托瓦斯编写了系统内核，称为 Linux，把所有软件和硬件连接起来。Linux 内核为 GNU 工程画上了完满的句号。

Richard Stallman 说，Linux 并不能代表整个操作系统。Linux 只是内核，整个系统还包含数以百计的软件工具和实用程序，大多是由 GNU 黑客们完成的。他认为，整个操作系统称为 GNU/Linux 比较合适。

Richard Stallman 认为，在 Copyleft 时代，软件公司可以靠服务和训练赚钱。如果你公司没有人会用源代码，你就得请位程序员，帮你修改由 FSF 得来的 Copyleft 程序；你不必怕你出钱所改的程序会流传到另一家公司，因为那家公司也许会为这软件改头换面，帮它抓虫，或修改，或添加些新功能。而在任意拷贝的情况下，你也因而受惠。

所以程序员绝对饿不死，仍会像现在高价软件的时代一样，有许多“服务”的大钱可赚，只不过不可能象盖茨这样积聚起世界第一的巨额财富。而 GNU 的软件也能使写程序的人更具生产力，因为他不必凡事都从零做起，可根据已有的软件来改进。所以 Richard Stallman 希望，有一天软件业者不是靠目前的“Copyright”版权法，迫使客户花费巨额资金购买软件，而是依仗提供服务(如技术支援、训练)来获取应得的报酬，这种报酬可能会比一般人高，但是绝对不可能为一个小公司培养出几百个百万富翁。简而言之，未来软件业的基本准则就是“资源免费，服务收费”。

近几年，随着 Linux 的迅速崛起，再也没有人对自由软件的全新商业模式表示怀疑。在 Richard Stallman 思想的指导下，自由软件已经成功地步入市场主流，占据了市场实地。毕竟，在商业横流的今天，思想在贬值。自由软件也只有在夺取市场政权后，才能真正确立自己的实力地位，促使整个软件业模式发生巨变。

孤独是思想家的归宿

目前发展的势头表明，完全站在用户一边的自由软件不可抵挡。它面临的唯一敌人还是自由软件领域内部的分裂和争斗。除了市场原有垄断者外，这是任何人都不希望看到的。

Richard Stallman 总是风尘仆仆，行囊相随，四处布道。他带着一台笔记本电脑，但这不是他个人的，而是属于自由软件基金（FSF）。其实，Richard Stallman 从来就没有拥有过一台自己的计算机。也从来只用自由软件（当然他从来没有用过 Windows ）。而且，他也没有自己的汽车、电视和房产。这位46岁的单身汉节俭地居住在一间租来的房子里。已有15年了，没有领取过一个月的正式工资。因为他的工作就是使软件获得自由。在商欲横流的今天，人们更愿意追逐财富，而不是贫寒的 Richard Stallman 的高尚思想。因此不足为怪，连自由软件团体内的许多人也开始离他而去。

随着自由软件迅速崛起，影响力大增。Richard Stallman 毫不妥协的个性和思想使其在自由软件内部也越来越成为争议人物。随着 Apache Web服务器和 GNU/Linux 操作系统的日渐流行，新一代黑客们受到鼓舞，纷纷投入商业领域，越来越多的人加盟自由软件，他们是一类

全新的黑客：一方面呼应自由软件的精神，一方面又积极拥抱商业世界。他们鼓吹自己能够创建比专有软件更稳定更灵活更少“臭虫”的软件产品，同时又积极捕捉每一个商机。

于是，“持不同政见者”将自由软件的标签改成了“开源软件”。看起来好象两者兼容，但Richard Stallman认为，两者最大的区别就是后者将自由精神放在首位。但是对商人来说，自由（free）与免费（free）是同一个词。与 Richard Stallman 布道“自由精神”不同，他们更愿意谈论实际问题。为了避免纷涌而来的投资者被 Richard Stallman“吓跑”，他们还得联起手来，将 Richard Stallman 屏蔽起来。

Richard Stallman 不喜欢实用主义的辩词，他宣称即使一个自由软件不是他最佳的解决方案，他也愿意作为首选。对 Richard Stallman 来说，自由是基本的道德美德：所有的计算机用户都应该享有相互协作、共享，以及拷贝和交换源代码的自由。

他的态度使一些“开放源代码”社区派系的人感到不安。实际上，他们标出“开放源代码”新旗号的目的就是要与激进分子 Richard Stallman 划清界线。不可否认，正是 Richard Stallman 领导了自由软件运动，正是他自己开发的千万行程序代码使如今的“Linux”成为一个整体，并走向成功。但对于新一代的人， Richard Stallman 是一种困窘，是一种障碍，更是一个捣乱分子。必须将他不惜代价地推入密室，以免吓跑投资者。

现实主义与理想主义之争

毫无疑问，无论是自由软件运动还是开放源代码，都来源他15年前开始的努力，这肯定是计算机历史上最脍炙人口的传奇故事。对于目前的成功， Richard Stallman 感到非常高兴，但也有更多的焦虑。因为他感到自己明显被排斥在外。“有人极力想改变历史，否定我在这场运动中的地位”对于人们用 Linux 来指代整个操作系统， Richard Stallman 十分痛心，他说正确的用词应是“GUN/Linux”。 Richard Stallman 承认托瓦斯的贡献很关键，是他完成了GUN/Linux 的内核。但是 Richard Stallman 估算，内核只占整个系统的3%，相比之下，GUN 项目贡献了30%的代码，其余67%的代码来源于其他方面。但令他欣慰的是，GNU 的一些原则仍在起作用。他认为这种原则不仅使软件开发更显活力，更能生产出优质软件，还认识到这本身是一种行为准则。

批评者认为， Richard Stallman 极力维护 GNU 的遗产，是沉湎于这场运动的枝枝末末，对整个自由软件都是有害的。对大多数开放源代码倡导者来说，颠覆微软才是主要的斗争方向。

“我关心的是精神，是 GUN 项目内在的哲学。这种哲学就是它存在的理由，那就是自由软件不仅仅是为了方便，也不仅仅是为了可靠。真正重要的是自由，协作的自由。我不关心某个人或公司。因此我认为单纯与微软作战偏离了这个运动的方向。”

Richard Stallman 的拥戴者还有，但是 Richard Stallman 的大多数主张还是被人们忽略了。如今只有“Linux”充斥着媒体的标题，而背后的思想开始逐渐隐去。

Richard Stallman 被邀请参加在硅谷湾区举办的“开放源代码开发者日”。这个1998年8月21日开幕的活动是程序员和自由软件热心家的大集会，由计算机图书出版公司 OReilly 联合公司组织的。而在4月份组织的“自由软件高峰会议”上， Richard Stallman 没有被邀请。结果招致了广泛的批评，使得组织者再也不敢“忽视”他了。但是组织者告诫他，要他以“维护大局”为重，让他在有分歧的地方免开尊口。

不足为奇， Richard Stallman 让许多自由软件的同行们十分棘手。他这个人不可控制，不可预知也不可能被改变。是这些非同寻常的素质促成了自由软件的兴起。但是随着自由软件前景大开，江山指日可待，这些个性开始被视为障碍。 Richard Stallman 的狂热和信仰保证了自由软件的成长，但如今人们认为他是多余的。在他们眼里，现在剔除了 Richard Stallman ，自由软件不但不会受阻，反而会更顺畅。

但不容置疑的是，正是 Richard Stallman 的思想成为自由软件运行的力量源泉。 GNU/Linux 系统的确比 Windows 系统问题少，Apache 也是 Web 服务器的更好选择。但这只是问题的一个侧面，只有把实用和理想结合起来，自由软件运动才令人信服，才能激发人们的热情和献身精神。无论如何， Richard Stallman 仍然是有名气的，仍然受到尊重，仍然被认为是自由软件运动的核心人物。 Richard Stallman 在计算机领域的重要地位不容动摇。为了使自由软件商业化而牺牲 Richard Stallman ，那就可能会失去这场重大革命的灵魂和方向。后记

Richard Stallman 依旧没钱、没势，连原先的许多信徒都被分化而去。这场运动给他带来的唯一收获可能就是：无论 Richard Stallman 走到哪里，都会有人乐意借给他计算机，使他能及时查看电子邮件。他还是那样不修边幅，无所顾忌。但是与当年执着相比，他的精神状态开始呈现一种新的焦虑和紊乱，而这一切正是他创造的自由软件的成功，施加给他的。而且可以肯定， Richard Stallman 必将越来越被自由软件成功的浪潮所淹没。

未来难以预测，唯一可以肯定的是： Richard Stallman 自己引燃的这场革命已经完全超越了他的驾驭范围。这也是许多思想家共同的命运。

其实，人类导演的故事总是一模一样的。

第 48 章 Linux

目录

Linux 诞生和发展

作者:不详

Linux 操作系统是UNIX 操作系统的一种克隆系统。它诞生于1991 年的10 月5 日（这是第一次正式向外公布的时间）。以后借助于Internet 网络，并经过全世界各地计算机爱好者的共同努力下，现已成为今天世界上使用最多的一种UNIX 类操作系统，并且使用人数还在迅猛增长。Linux 操作系统的诞生、发展和成长过程始终依赖着以下五个重要支柱：UNIX 操作系统、MINIX 操作系统、GNU 计划、POSIX 标准和Internet 网络。

下面主要根据这五个基本线索来追寻一下Linux 的开发历程，它的酝酿过程，最初的发展经历。首先分别介绍其中的四个基本要素(UNIX、MINIX、GNU 和POSIX，Internet 的重要性显而易见，所以不再赘述)，然后根据Linux 的创始人Linus Toravlds 从对计算机感兴趣而自学计算机知识，到心里开始酝酿编制一个自己的操作系统，到最初Linux 内核0.01 版公布，以及从此如何艰难地一步一个脚印地在全世界hacker 的帮助下最后推出比较完善的1.0 版本这段时间的发展经过，也即对Linux 的早期发展历史进行详细介绍。

UNIX 操作系统的诞生

Linux 操作系统是UNIX 操作系统的一个克隆版本。UNIX 操作系统是美国贝尔实验室的Ken.Thompson和Dennis Ritchie 于1969 年夏在DEC PDP-7 小型计算机上开发的一个分时操作系统。当时Ken Thompson 为了能在闲置不用的PDP-7 计算机上运行他非常喜欢的星际旅行（Space travel）游戏，在1969 年夏天乘他夫人回家乡加利福尼亚渡假期间，在一个月内开发出了unix 操作系统的原型。当时使用的是BCPL 语言（基本组合编程语言），后经Dennis Ritchie 于1972 年用移植性很强的C 语言进行了改写，使得UNIX 系统在大专院校得到了推广。

MINIX 操作系统

MINIX 系统是由Andrew S. Tanenbaum（AST）开发的。AST 是在荷兰Amsterdam 的Vrije 大学数学与计算机科学系统工作，是ACM 和IEEE 的资深会员(全世界也只有很少人是两会的资深会员)。共发表了100 多篇文章，5 本计算机书籍。AST 虽出生在美国纽约，但是是荷兰侨民(1914 年他的祖辈来到美国)。他在纽约上的中学、M.I.T上的大学、加洲大学Berkeley 分校念的博士学位。由于读博士后的缘故，他来到了家乡荷兰。从此就与家乡一直有来往。后来就在Vrije 大学开始教书、带研究生了。荷兰首都Amsterdam 是个常年阴雨绵绵的城市，而对于AST 来说，这最好不过了，因为这样他就可以待在家里摆弄他的计算机了。MINIX 是他1987年编制的，主要用于学生学习操作系统原理。到91年时版本是1.5。目前主要有两个版本在使用： 1.5 版和2.0 版，当时该操作系统在大学使用是免费的，但其它用途不是，当然目前都已经是免费的，可以从许多FTP上下载。

对于Linux 系统，他表示对其开发者Linus 的称赞。但他认为Linux 的发展有很大原因是因为他为了保持minix 的小型化，能让学生在一个学期内就能学完，而没有接纳全世界许多人对Minix 的扩展要求。因此这激发了Linus 编写Linux。Linus 正好抓住了这个好时机。

作为一个操作系统，MINIX 并不是优秀者，但它同时提供了用C 语言和汇编语言写的系统源代码。这是第一次使得有抱负的程序员或hacker 能够阅读操作系统的源代码，在当时这种源代码是软件商一直小心地守护着的。

GNU 计划

GNU 计划和自由软件基金会(the Free Software Foundation - FSF)是由Richard M. Stallman 于1984 年一手创办的。旨在开发一个类似 Unix、并且是自由软件的完整操作系统：GNU 系统。（GNU 是"GNU's Not Unix"的递归缩写，它的发音为"guh-NEW"。）各种使用linux 作为核心的GNU 操作系统正在被广泛的使用。虽然这些系统通常被称作"Linux"，但是严格地说，它们应该被称为GNU/Linux 系统。

到上世纪90 年代初，GNU 项目已经开发出许多高质量的免费软件，其中包括有名的emacs 编辑系统、bash shell 程序、gcc 系列编译程序、gdb 调试程序等等。这些软件为Linux 操作系统的开发创造了一个合适的环境，是Linux 能够诞生的基础之一。以至于目前许多人都将Linux 操作系统称为"GNU/Linux"操作系统。

POSIX 标准

POSIX(Portable Operating System Interface for Computing Systems)是由IEEE 和ISO/IEC 开发的一簇标准。该标准是基于现有的UNIX 实践和经验，描述了操作系统的调用服务接口，用于保证编制的应用程序可以在源代码一级上在多种操作系统上移植运行。它是在1980 年早期一个UNIX 用户组(usr/group)的早期工作的基础上取得的。该UNIX 用户组原来试图将AT&T 的系统V 和Berkeley CSRG的BSD 系统的调用接口之间的区别重新调和集成，从而于1984 年产生了/usr/group 标准。1985 年，IEEE操作系统技术委员会标准小组委员会（TCOS-SS）开始在ANSI 的支持下责成IEEE 标准委员会制定有关程序源代码可移植性操作系统服务接口正式标准。到了1986 年4 月，IEEE 就制定出了试用标准。第一个正式标准是在1988 年9 月份批准的（IEEE 1003.1-1988），也既以后经常提到的POSIX.1 标准。

1989 年POSIX 的工作被转移至ISO/IEC 社团，并由15 工作组继续将其制定成ISO 标准。到1990 年，POSIX.1 与已经通过的C 语言标准联合，正式批准为IEEE 1003.1-1990（也是ANSI 标准）和ISO/IEC 9945-1:1990 标准。

POSIX.1 仅规定了系统服务应用程序编程接口（API），仅概括了基本的系统服务标准，因此期望对系统的其它功能也制定出标准。这样IEEE POSIX 的工作就开始展开了。在1990 年，刚开始有十个批准的计划在进行，有近300 多人参加每季度为期一周的会议。着手的工作有命令与工具标准(POSIX.2)、测试方法标准（POSIX.3）、实时API（POSIX.4）等。到了1990 年上半年已经有25 个计划在进行，并且有16 个工作组参与了进来。与此同时，还有一些组织也在制定类似的标准，如X/Open，AT&T，OSF 等。

在90 年代初，POSIX 标准的制定正处在最后投票敲定的时候，那是1991-1993 年间。此时正是Linux刚刚起步的时候，这个UNIX 标准为Linux 提供了极为重要的信息，使得Linux 的能够在标准的指导下进行开发，能够与绝大多数UNIX 系统兼容。在最初的Linux 内核代码中(0.01 版、0.11 版)就已经为Linux与POSIX 标准的兼容做好了准备工作。在0.01 版的内核/include/unistd.h 文件中就已经定义了几个有关POSXI 标准要求的常数符号，并且在注释中就写到"ok，这也许是个玩笑，但我正在着手研究它呢"。

1991 年7 月3 日在comp.os.minix 上发布的post 上就已经提到了正在搜集POSIX 的资料。(当然此时还不存在Linux 这个名称，当时Linus 的脑子里想的可能是FREAX ，FREAX 的英文含义是怪诞的、怪物、异想天开等)。其中透露了他正在进行Linux 系统的开发，并且在Linux 最初的时候已经想到要实现与POSIX（UNIX 的国际标准）的兼容问题了。

Linux 操作系统的诞生

1981 年IBM 公司推出享誉全球的微型计算机IBM PC。在1981-1991 年间，MS-DOS 操作系统一直是微型计算机上操作系统的主宰。此时计算机硬件价格虽然逐年下降，但软件价格仍然是居高不下。当时Apple 的MACs 操作系统可以说是性能最好的，但是其天价没人能够轻易靠近。

当时的另一个计算机技术阵营是Unix世界。但是Unix操作系统就不仅是价格贵的问题了。为了寻求高利率，Unix经销商将价格抬得极高，PC小用户就根本不能靠近它。曾经一度受到Bell Labs的许可而可以在大学中用于教学的UNIX源代码一直被小心地守卫着不需公开。对于广大的PC用户，软件行业的大型供应商始终没有给出有效的解决该问题的手段。正在此时，出现了MINIX操作系统，并有一本详细的书本描述它的设计实现原理。由于AST的书写的非常详细，并且叙述有条有理，几乎全世界的计算机爱好者都在看这本书以理解操作系统的工作原理。其中也包括Linux系统的创始者Linus Benedict Torvalds。当时(1991年)，Linus Benedict Torvalds 是赫尔辛基大学计算机科学系的二年级学生，也是一个自学hacker。这个21 岁的芬兰年轻人喜欢鼓捣计算机，测试计算机的能力和限制。但当时缺乏的是一个专业级的操作系统。MINIX 虽然很好，但只是一个用于教学目的简单操作系统，而不是一个强有力的实用操作系统。

到1991 年，GNU 计划已经开发出了许多工具软件。最受期盼的Gnu C 编译器已经出现，但还没有开发出免费的GNU 操作系统。即使是MINIX 也开始有了版权，需要购买才能得到源代码。而GNU 的操作系统HURD 一直在开发之中，但并不能在几年内完成。对于Linus 来说，已经不能等待了。从1991 年4 月份起，他开始酝酿并着手编制自己的操作系统。刚开始，他的目的很简单，只是为了学习Intel 386 体系结构保护模式运行方式下的编程技术。但后来Linux 的发展却完全改变了初衷。

1991 年初，Linux 开始在一台386sx 兼容微机上学习minix 操作系统。通过学习，他逐渐不能满足minix 系统的现有性能，并开始酝酿开发一个新的免费操作系统。根据Linus 在comp.os.minix 新闻组上发布的消息，我们可以知道他逐步从学习minix 系统到开发自己的Linux 的过程。

Linus 第1 次向comp.os.minix 投递消息是在1991 年3 月29 日。题目是"gcc on minix-386 doesn't optimize",是有关gcc 编译器在minix-386 上运行的优化问题，由此可知，Linus 在1991 年的初期已经开始深入研究了minix 系统，并在这段时间有了改进minix 操作系统的思想，而且在进一步学习minix系统中，逐步演变为想自己重新设计一个基于Intel 80386 体系结构的新操作系统。他在回答有人提出minix 上的一个问题时，所说的第一句话是"阅读源代码"（"RTFSC (Read the Fucking Source Code :-)"）。他认为答案就在源程序中。这也说明了对于学习系统软件来说，你不光需要懂得系统的工作基本原理，还需要结合实际系统，学习实际系统的实现方法。因为理论毕竟是理论，其中省略了许多枝节，而这些枝节问题虽然没有太多的理论含量，但却是一个系统必要的组成部分，就像麻雀身上的一根羽毛。

从1991 年的4 月份开始，Linus 几乎花了全部时间研究386-minix 系统(hack the kernel)，并且尝试着移植GNU 的软件到该系统上(GNU gcc、bash、gdb 等)。并于4 月13 日在comp.os.minix 上发布说自己已经成功地将bash 移植到了minix 上，而且已经爱不释手、不能离开这个shell 软件了。

第一个与Linux 有关的消息是在1991 年7 月3 日在comp.os.minix 上发布的(当然此时还不存在Linux 这个名称，当时Linus 的脑子里想的可能是FREAX ，FREAX 的英文含义是怪诞的、怪物、异想天开等)。其中透露了他正在进行Linux 系统的开发，并且在Linux 最初的时候已经想到要实现与POSIX（UNIX 的国际标准）的兼容问题了。

在Linus 的下一发布的消息中(1991 年8 月25 日 comp.os.minix)，他向所有minix 用户询问"What would you like to see in minix?"("你最想在minix 中见到什么？")，在该消息中他首次透露出正在开发一个(免费的)386(486)操作系统，并且说只是兴趣而已，代码不会很大，也不会象GNU 的那样专业。开发免费操作系统这个想法从4 月份就开始酝酿了，希望大家反馈一些对于minix 系统中喜欢那些特色不喜欢什么等信息，由于实际的和其它一些原因，新开发的系统刚开始与minix 很象（并且使用了minix 的文件系统）。并且已经成功地将bash(1.08 版)和gcc(1.40 版)移植到了新系统上，而且在过几个月就可以实用了。

最后，Linus 申明他开发的操作系统没有使用一行minix 的源代码；而且由于使用了386 的任务切换特性，所以该操作系统不好移植（没有可移植性），并且只能使用AT 硬盘。对于Linux 的移植性问题，Linus 当时并没有考虑。但是目前Linux 几乎可以运行在任何一种硬件体系结构上。

到了1991 年的10 月5 日，Linus 在comp.os.minix 新闻组上发布消息，正式向外宣布Linux 内核系统的诞生（Free minix-like kernel sources for 386-AT）。这段消息可以称为Linux 的诞生宣言，并且一直广为流传。因此10 月5 日对Linux 社区来说是一个特殊的日子，许多后来Linux 的新版本发布时都选择了这个日子。所以RedHat 公司选择这个日子发布它的新系统也不是偶然的。

Linux 操作系统版本的变迁

- 0.00 (1991.2-4?) 两个进程分别显示AAA BBB
- 0.01 (1991.9?)第一个正式向外公布的Linux 内核版本。
- 0.02 (1991.10.5)该版本以及0.03 版是内部版本，目前已经无法找到。
- 0.03 (1991.10.5)
- 0.10 (1991.10)由Ted Ts'o 发布的Linux 内核版本。
- 0.11 (1991.12.8)基本可以正常运行的内核版本。
- 0.12 (1992.1.15)主要加入对数学协处理器的软件模拟程序。
- 0.95 (0.13) (1992.3.8) 开始加入虚拟文件系统思想的内核版本。
- 0.96 (1992.5.12)开始加入网络支持和虚拟文件系统VFS。
- 0.97 (1992.8.1)
- 0.98 (1992.9.29)
- 0.99 (1992.12.13)
- 1.0 (1994.3.14)
- 1.20 (1995.3.7)
- 2.0 (1996.2.9)
- 2.20 (1999.1.26)
- 2.40 (2001.1.4)
- 2.60 (2003.12.17)

将Linux 系统0.13 版内核直接改称0.95 版，Linus 的意思是让大家不要觉得离1.0 版还很遥远。同时，从0.95 版开始，对内核的许多改进之处(补丁程序的提供)均以其他人为主了，而Linus 的主要任务开始变成对内核的维护和决定是否采用某个补丁程序。

Linux 名称的来由

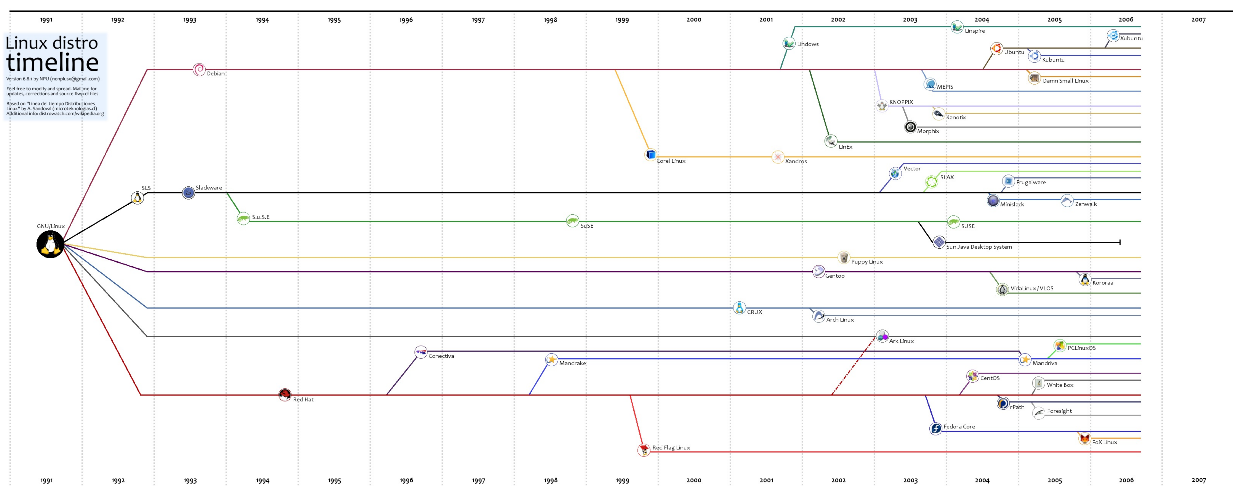
Linux 操作系统刚开始时并没有被称作Linux，Linus 给他的操作系统取名为FREAX，其英文含义是怪诞的、怪物、异想天开等意思。在他将新的操作系统上载到ftp.funet.fi 服务器上时，管理员Ari Lemke很不喜欢这个名称。他认为既然是Linus 的操作系统就取其谐音Linux 作为该操作系统的目录吧，于是Linux 这个名称就开始流传下来。

在Linus 的自传《Just for Fun》一书中，Linus 解释说："坦白地说，我从来没有想到过要用Linux 这个名称发布这个操作系统，因为这个名字有些太自负了。而我为最终发布版准备的是什么名字呢？Freax。实际上，内核代码中某些早期的Makefile - 用于描述如何编译源代码的文件 - 文件中就已经包含有"Freax"这个名字了，大约存在了半年左右。但其实这也没什么关系，在当时还不需要一个名字，因为我还没有向任何人发布过内核代码。而Ari Lemke，他坚持要用自己的方式将内核代码放到ftp 站点上，并且非常不喜欢Freax 这个名字。他坚持要用现在这个名字(Linux)，我承认当时我并没有跟他多争论。但这都是他取的名字。所以我可以光明正大地说我并不自负，或者部分坦白地说我并没有本位主义思想。但我想好吧，这也是个好名字，而且以后为这事我总能说服别人，就象我现在做的这样。"-- Linus Torvalds《Just for fun》第84-88 页。

通过上述说明，我们可以对上述Linux 的五大支柱归纳如下：

- UNIX 操作系统 -- UNIX 于1969 年诞生在Bell 实验室。Linux 就是UNIX 的一种克隆系统。UNIX的重要性就不用多说了。
- MINIX 操作系统 -- Minix 操作系统也是UNIX 的一种克隆系统，它于1987 年由著名计算机教授Andrew S. Tanenbaum 开发完成。由于MINIX 系统的出现并且提供源代码(只能免费用于大学内)在全世界的大学中刮起了学习UNIX 系统旋风。Linux 刚开始就是参照Minix 系统于1991 年才开始开发。
- GNU 计划-- 开发Linux 操作系统，以及Linux 上所用大多数软件基本上都出自GNU 计划。Linux只是操作系统的一个内核，没有GNU 软件环境(比如说bash shell)，则Linux 将寸步难行。
- POSIX 标准 -- 该标准在推动Linux 操作系统以后朝着正规路上发展起着重要的作用。是Linux前进的灯塔。
- INTERNET -- 如果没有Intenet 网，没有遍布全世界的无数计算机骇客的无私奉献，那么Linux最多只能发展到0.13(0.95)版的水平。

附录：Linux 发行版分支图


1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
Linux distro
timeline
Version 6.8.1 by NPU (nonplusx@gmail.com)
Feel free to modify and spread. Mail me for updates, corrections and source flw/xcf files
Based on "Linea del tiempo Distribuciones Linux" by A. Sandoval (microteknologias.cl)
Additional info: distrowatch.com/wikipedia.org
GNU/Linux
Debian
SLS
Slackware
S.u.S.E
SuSE
Red Hat
Conectiva
Mandrake
Red Flag Linux
Corel Linux
Xandros
Lindows
Linspire
Ubuntu
Kubuntu
Xubuntu
Damn Small Linux
MEPIS
KNOPPIX
Kanotix
Morphix
LinEx
Vector
SLAX
Frugalware
Minislack
Zenwalk
SUSE
Sun Java Desktop System
Puppy Linux
Gentoo
VidaLinux / VLOS
Kororaa
CRUX
Arch Linux
Ark Linux
PCLinuxOS
Mandriva
CentOS
White Box
rPath
Foresight
Fedora Core
FoX Linux

第 49 章 GNOME与KDE的战争

目录

作者:不详

虽然在商业方面存在竞争，GNOME与KDE两大阵营的开发者关系并没有变得更糟，相反他们都意识到支持对方的重要性—如果KDE和GNOME无法实现应用程序的共享，那不仅是巨大的资源浪费，而且将导致Linux出现根本上的分裂。

KDE 与GNOME是目前Linux/UNIX系统最流行的图形操作环境。从上个世纪九十年代中期至今，KDE和GNOME都经历了将近十年的漫漫历程，两者也都从最初的设计粗糙、功能简陋发展到相对完善的阶段，可用性逼近Windows系统。图形环境的成熟也为Linux的推广起到至关重要的作用，尽管 Linux以内核健壮、节省资源和高质量代码著称，但缺乏出色的图形环境让它一直难以在桌面领域有所作为，导致Linux桌面应用一直处于低潮。如果大家还有印象，一定会记得1999-2001年间Linux发展如火如荼，当时国内涌现出大量的Linux发行版厂商，但当用户发现Linux距离实用化还有十万八千里的时候，Linux热潮迅速冷却。业界也对此一度灰心失望，其中一部分厂商因无法盈利迅速销声匿迹，另一部分厂商则不约而同将重点放在服务器市场—与桌面市场形成鲜明对比的是，Linux以稳定可靠和低成本的优势在服务器领域获得了巨大的成功。

在一些Linux厂商放弃桌面化努力的同时，国际开源社群却不断发展壮大，自由的理念吸引越来越多一流的程序员参与。与商业模式不同，自由软件程序员在开始时都只是利用业余时间开发自己感兴趣的东西，并将其自由公开，这是一种不折不扣的贡献行为。尽管开发进度缓慢，但认同自由软件理念的开发者越来越多，一个个开源项目逐渐发展壮大。

在此期间一个被人忽视的重大事件就是商业巨头也积极参与进来，IBM、RedHat、SuSE、Ximian、 Novell、SUN、HP等商业公司都直接介入各个开源项目，这些企业或者是将自身的成果免费提供给开源社群，或者直接派遣程序员参与项目的实际开发工作，例如SuSE（现已为Novell收购）在KDE项目上做了大量的工作，RedHat、Ximian（现已为Novell收购）则全程参与Gnome 项目，IBM为Linux提供了大量的基础性代码，是推进Linux进入服务器领域的主要贡献者，SUN公司则将StarOffice赠送给开源社群，并资助成立著名的OpenOffice.org项目。这样，大量的自由软件程序员都可以从各个项目的基金会中领到薪水。在这一阶段，开源项目摆脱了程序员业余开发的模式，而由高水平的专职程序员主导，这也成为各个自由软

件项目的标准协作模式。与商业软件公司不同，自由软件项目的参与者都是首先为个人兴趣而工作，他们的共同目标都是拿出品质最好的软件，在协作模式稳定成形之后，各个软件就进入到发展的快行道。进入2005年后，这些项目基本上都获得了丰硕的成果，其中最突出的代表就是Firefox浏览器的成功，而作为两大图形环境，KDE和GNOME分别发展到3.5和2.12版本，两者的可用性完全可以媲美Windows。更重要的是，开源社群的发展壮大为这些项目的未来发展奠定了坚实的基础：KDE项目将超越Windows作为自己的目标，力量更强大的GNOME项目更是将开发目标定在超越Mac OS X的Aqua图形环境；Firefox则计划运用GPU的硬件资源来渲染图像，达到大幅度提高速度的目的；OpenOffice.org在努力提升品质的同时奠定了开放文档格式标准。除了上述主要项目之外，我们也看到如Mplayer播放器、Xine播放器、Thunderbird邮件客户端、SCIM输入平台等其他开源项目也在快速发展成熟之中，且几乎每一天都有新的项目在诞生。有意思的是，除了涉及到软件开发外，还出现了为Linux设计视觉界面的开放协作项目，全球各地有着共同目标的艺术家通过互联网组织到一起，共同为Linux系统设计一流的视觉界面、系统图标，而所有的自由软件程序员都有一个共同的目标，那就是开发出一流水准的软件提供给大众使用。这种基于挑战自我、带有浓烈精神色彩的软件开发模式成为商业软件之外的另外一极。现在，微软面对的并不是那些只在业余时间鼓捣代码的程序员，而是分布在全球各地、数量庞大、且拥有一流技术水平的开发者，这些开发者被有效地组织起来，形成一个个有序的协作团队，大量实力雄厚的商业公司在背后提供支持。虽然今天的Linux系统还无法在桌面领域被广为接纳，但只需要两、三年时间，高速进化的Linux平台将可达到全面进军桌面的水准，也正是看到其中的机会，Novell、RedHat等重量级Linux企业都不断在技术和市场推广方面加大投入，Linux 桌面化近在咫尺。

在介绍完必要的背景之后，我们将进入关于KDE与GNOME的技术专题。如果你是刚刚接触Linux的新手，一定会对KDE和GNOME感到困惑不已—为何会有两个功能重复、操作习惯迥异的图形环境？这不仅麻烦也耗费开发者精力。通过本文，你将获得清晰的答案。而更重要的是，我们将在本文中向大家介绍 KDE与GNOME的实际水平、各自的优点和未来发展趋势。如果你对Linux桌面应用有些兴趣，那么未来的 KDE/GNOME一定会让你感到震惊不已。

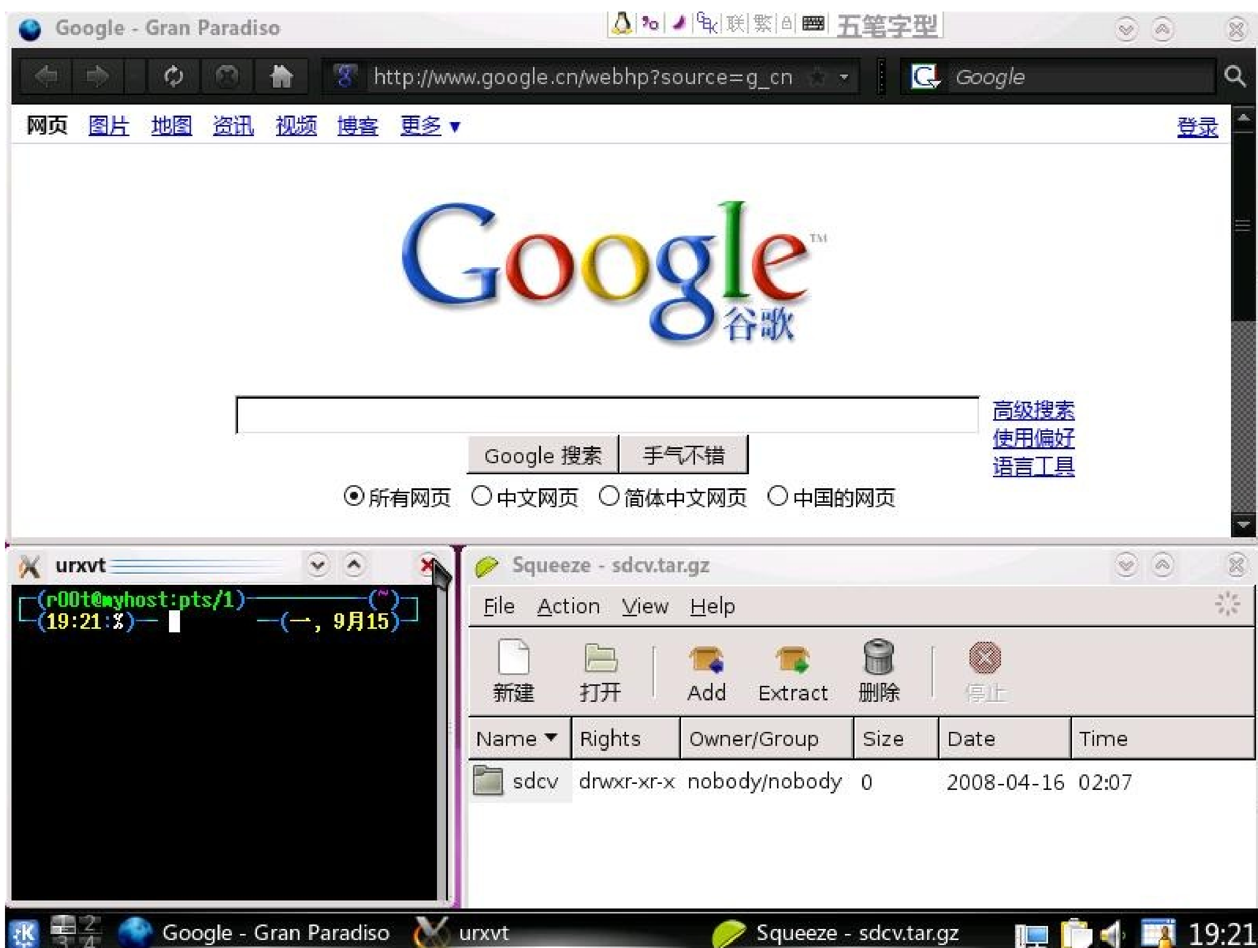
X Window 打造桌面环境

在介绍KDE和Gnome之前，我们有必要先来介绍UNIX/Linux图形环境的概念。对一个习惯Windows的用户来说，要正确理解UNIX /Linux的图形环境可能颇为困难，因为它与纯图形化Windows并没有多少共同点。Linux实际上是以UNIX为模板的，它继承了UNIX内核设计精简、高度健壮的特点，无论系统结构还是操作方式也都与UNIX无异。简单点说，你可以将Linux看成是UNIX类系统中的一个特殊版本。我们知道，微软Windows在早期只是一个基于DOS的应用程序，用户必须首先进入DOS后再启动Windows进程，而从Windows 95开始，微软将图形界面作为默认，命令行界面只有在需要的情况下才开启，后来的Windows98/Me实际上也都隶属于该体系。但在 Windows2000之后，DOS被彻底清除，Windows成为一个完全图形化的操作系统。但UNIX/Linux与之不同，强大的命令行界面始终是它们的基础，在上个世纪八十年代中期，图形界面风潮席卷操作系统业界，麻省理工学院（MIT）也在1984年与当时的DEC公司合作，致力于在UNIX系统上开发一个分散式的视窗环境，这便是大名鼎鼎的“X

Window System”项目。不过，X Window（请注意不是XWindows）并不是一个直接的图形操作环境，而是作为图形环境与UNIX系统内核沟通的中间桥梁，任何厂商都可以在 XWindow基础上开发出不同的GUI图形环境。MIT和DEC的目的只在于为UNIX系统设计一套简单的图形框架，以使UNIX工作站的屏幕上可显示更多的命令，对于GUI的精美程度和易用程度并不讲究，毕竟那时候能够熟练操作UNIX的都是些习惯命令行的高手，根本不在乎GUI存在与否。1986 年，MIT正式发行X Window，此后它便成为UNIX的标准视窗环境。紧接着，全力负责发展该项目的X协会成立，XWindow进入了新阶段。与此同步，许多UNIX厂商也在X Window原型上开发适合自己的UNIXGUI视窗环境，其中比较著名的有SUN与AT&T联手开发的“Open Look”、IBM主导下的OSF（Open SoftwareFoundation，开放软件基金会）开发出的“Motif”。而一些爱好者则成立了非营利的XFree86组织，致力于在X86系统上开发XWindow，这套免费且功能完整的XWindow很快就进入了商用UNIX系统中，且被移植到多种硬件平台上，后来的Linux也直接从该项目中获益。当然，这些早期的XWindow环境都设计得很简单，许多GUI元素模仿于微软的Windows，但XWindow拥有一个小小的创新：当鼠标指针移动到某个窗口时，该窗口会被自动激活，用户无需点击便能够直接输入，简化了用户操作—这个特性在后来的KDE和Gnome中也都得到完整的继承。

由于必须以UNIX系统作为基础，XWindow注定只能成为UNIX上的一个应用，而不可能与操作系统内核高度整合，这就使得基于XWindow的图形环境不可能有很高的运行效率，但它的优点在于拥有很强的设计灵活性和可移植性。X Window从逻辑上分为三层：最底层的XServer（X服务器）主要处理输入/输出信息并维护相关资源，它接受来自键盘、鼠标的操作并将它交给X Client（X客户端）作出反馈，而由XClient传来的输出信息也由它来负责输出；最外层的XClient则提供一个完整的GUI界面，负责与用户的直接交互（KDE、Gnome都是一个X Client），而衔接X Server与XClient的就是“X Protocol(X通讯协议)”、它的任务是充当这两者的沟通管道。尽管UNIX厂商采用相同的XWindow，但由于终端的X Client并不相同，这就导致不同UNIX产品搭配的GUI界面看起来非常不一样。

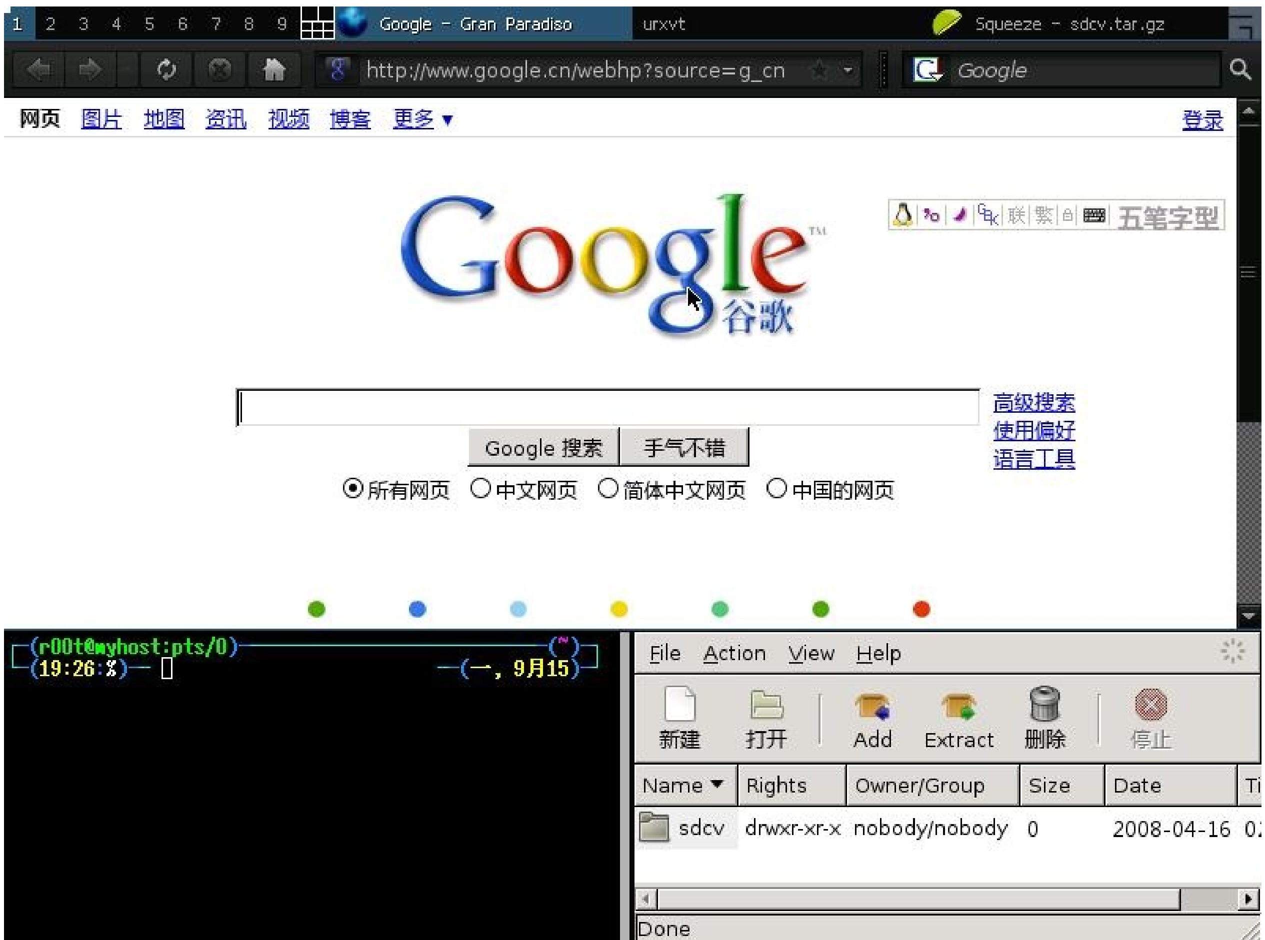
图 **49.1. X Window**系统架构示意图



KDE项目的发起

MIT的X Window推出之后就成为UNIX图形界面的标准，但在商业应用上分为两大流派：一派是以Sun公司领导的OpenLook阵营，一派是IBM/HP领导的OSF(Open SoftwareFoundation)的Motif，双方经过多年竞争之后，Motif最终获得领先地位。不过，Motif只是一个带有窗口管理器（Window-Manager）的图形界面库（Widget-Library），而非一个真正意义上的GUI界面。经过协商之后IBM/HP与SUN决定将Motif与Open Look整合，并在此基础上开发出一个名为“CDE(Common Desktop Environment)”的GUI作为UNIX的标准图形界面。遗憾的是，Motif/CDE和UNIX系统的价格都非常昂贵，而当时微软的 Windows发展速度惊人并率先在桌面市场占据垄断地位，CDE则一直停留在UNIX领域提供给root系统管理员使用，直到今天情况依然如此。

图 **49.2. KDE 1.0**尽管设计粗糙，但它奠定了整个**KDE**项目的基础。



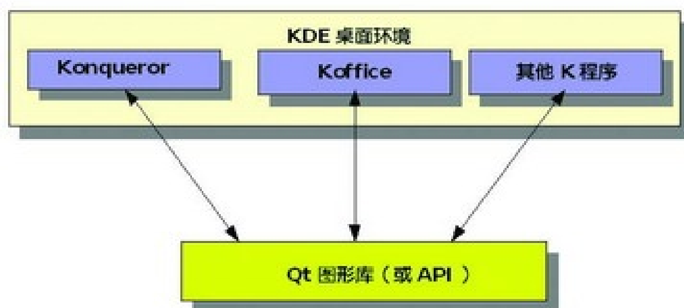
在上个世纪九十年代中期，以开源模式推进的Linux在开发者中已经拥有广泛的影响力。尽管X Window已经非常成熟，也有不少基于XWindow的图形界面程序，但它们不是未具备完整的图形操作功能就是价格高昂（如CDE），根本无法用于 Linux系统中。如果Linux要获得真正意义上的突破，一套完全免费、功能完善的GUI就非常必要。1996年10月，图形排版工具Lyx的开发者、一位名为MatthiasEttrich的德国人发起了KDE（Kool Desktop Environment）项目，与之前各种基于XWindow的图形程序不同的是，KDE并非针对系统管理员，它的用户群被锁定为普通的终端用户，MatthiasEttrich希望KDE能够包含用户日常应用所需要的所有应用程序组件，例如Web浏览器、电子邮件客户端、办公套件、图形图像处理软件等等，将UNIX/Linux彻底带到桌面。当然，KDE符合GPL规范，以免费和开放源代码的方式运行。

KDE项目发起后，迅速吸引了一大批高水平的自由软件开发者，这些开发者都希望KDE能够将Linux系统的强大能力与舒适直观的图形界面联结起来，创建最优秀的桌面操作系统。经过艰苦卓绝的共同努力，KDE 1.0终于在1998年的7月12日正式推出。以当时的水平来说，KDE1.0在技术上可圈可点，它较好的实现了预期的目标，各项功能初步具备，开发人员已经可以很好地使用它了。当然，对用户来说，KDE1.0远远比不上同时期的Windows 98来得平易近人，KDE 1.0中大量的Bug更是让人头疼。但对开发人员来说，KDE1.0的推出鼓舞人心，它证明了KDE项目开源协作的开发方式完全可行，开发者对未来充满信心。有必要提到的是，在KDE1.0版的开发过程中，SuSE、Caldera等Linux商业公司对该项目提供资金上的支持，在1999年，IBM、 Corel、RedHat、富士通-西门子等公司也纷纷对KDE项目提供资金和技术支持，自此KDE项目走上了快速发展阶段并长期保持着领先地位。但在 2004年之后，

GNOME不仅开始在技术上超越前者，也获得更多商业公司的广泛支持，KDE丧失主导地位，其原因就在于KDE选择在Qt平台的基础上开发，而Qt在版权方面的限制让许多商业公司望而却步。

Qt是一个跨平台的C++图形用户界面库，它是挪威TrollTech公司的产品。基本上，Qt同XWindow上的 Motif、Open Look、GTK等图形界面库和Windows平台上的MFC、OWL、VCL、ATL是同类型的东西，但Qt具有优良的跨平台特性（支持 Windows、Linux、各种UNIX、OS390和QNX等）、面向对象机制以及丰富的API，同时也可支持2D/3D渲染和OpenGLAPI。在当时的同类图形用户界面库产品中，Qt的功能最为强大，MatthiasEttrich在发起KDE项目时很自然选择了Qt作为开发基础，也正是得益于 Qt的完善性，KDE的开发进展颇为顺利，例如Netscape5.0在从Motif移植到Qt平台上仅仅花费了5天时间。这样，当KDE1.0正式发布时，外界看到的便是一个各项功能基本具备的GUI操作环境，且在后来的发展中，Qt/KDE一直都保持领先优势。有必要提到的是，TrollTech公司实质性参与了KDE项目，如前面提到Netscape 5.0的移植工作就是由TrollTech的程序员完成，而KDE工程的发起者、MatthiasEttrich本人也在1998年离开学术界加入 TrollTech，并一直担任该公司的软件开发部主管，因此TrollTech公司对于KDE项目拥有非常强的影响力（当然不能说绝对掌握，毕竟KDE 开发工作仍然是由自由程序员协作完成的）。我们前面提到，KDE采用GPL规范进行发行，但底层的基础Qt却是一个不遵循GPL的商业软件，这就给KDE 上了一道无形的枷锁并带来可能的法律风险。一大批自由程序员对KDE项目的决定深为不满，它们认为利用非自由软件开发违背了GPL的精神，于是这些GNU 的狂热信徒兵分两路：其中一部分人去制作Harmonny，试图重写出一套兼容Qt的替代品，这个项目虽然技术上相对简单，但却没有获得KDE项目的支持；另一路人马则决定重新开发一套名为“GNOME（GNU Network ObjectEnvironment）”的图形环境来替代KDE，一场因为思想分歧引发的GUI之战开始了。

图 **49.3. Qt**是整个**KDE**的基础，它采用双重授权。

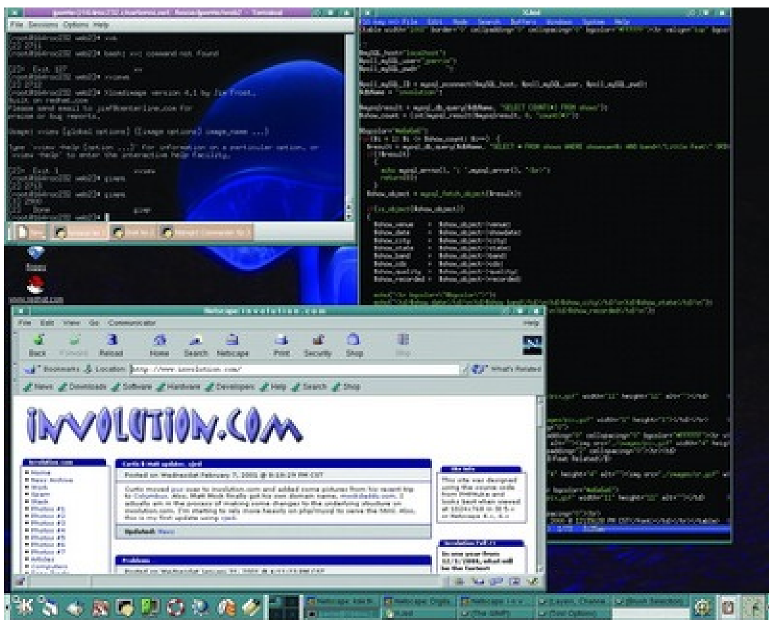


GNOME与KDE交替发展

GNOME项目于1997年8月发起，创始人是当时年仅26岁的墨西哥程序员Miguel Delcaza。关于GNOME的名称有一个非常有趣的典故：Miguel到微软公司应聘时对它的ActiveX/COMmodel颇有兴趣，GNOME（Network Object Model）的名称便从此而来。GNOME选择完全遵循GPL的GTK图形界面库为基础，因此我们也一般将GNOME和KDE两大阵营称为 GNOME/GTK和KDE/Qt。与Qt基于C++语言不同，GTK采用较传统的C语言，虽然

C语言不支持面向对象设计，看起来比较落后，但当时熟悉C 语言的开发者远远多于熟悉C++的开发者。加之GNOME/GTK完全遵循GPL版权公约，吸引了更多的自由程序员参与，但由于KDE先行一步，且基础占优势，一直都保持领先地位。1999年3月，GNOME 1.0在匆忙中推出，稳定性奇差无比，以至于许多人笑称GNOME 1.0还没有KDE 1.0Alpha稳定，而同期的KDE 1.1.2无论在稳定性还是功能上都远胜于GNOME，直到10月份推出的GNOME1.0.55版才较好解决了稳定性问题，给GNOME重新赢回声誉。由于思想分歧，当时GNOME的开发者与KDE的开发者在网络上吵得天翻地覆，几乎达到相互仇视的地步。但不管怎么说，GNOME都跌跌撞撞迈出了第一步，尽管那时KDE几乎是所有Linux发行版默认的桌面环境。

图 **49.4. KDE2.0**拥有丰富的应用软件，实力明显超过**GNOME**

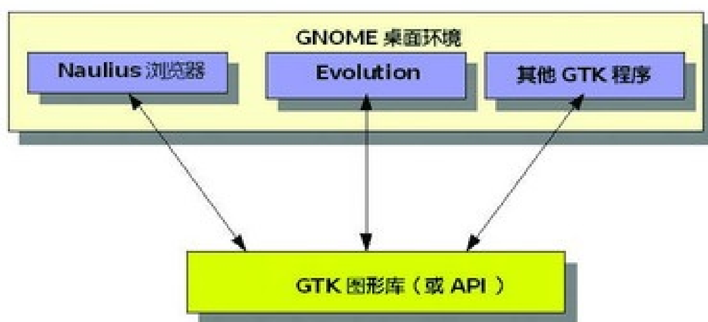


GNOME的转机来自于商业公司的支持。当时Linux业界的老大RedHat很不喜欢KDE/Qt的版权，在GNOME项目发起后RedHat立刻对其提供支持。为了促进GNOME的成熟，RedHat甚至专门派出几位全职程序员参与GNOME的开发工作，并在1998年1月与GNOME项目成员携手成立了RedHat高级开发实验室。1999年4月，Miguel与另一名GNOME项目的核心成员共同成立HelixCode公司为GNOME提供商业支持，这家公司后来更名为Ximian，它事实上就成为GNOME项目的母公司，GNOME平台上的Evolution邮件套件便出自该公司之手。进入 2000年之后，一系列重大事件接连发生，首先，一批从苹果公司出来的工程师成立Eazel公司，为GNOME设计用户界面和Nautilus（鹦鹉螺）文件管理器。同年8月，GNOME基金会在Sun 、RedHat、Eazel、HelixCode（Ximian）的共同努力下正式成立，该基金会负责GNOME项目的开发管理以及提供资金，Miguel本人则担任基金会的总裁。此时，GNOME获得许多重量级商业公司的支持，如惠普公司采用GNOME作为HP-UX系统的用户环境，SUN则宣布将StarOffice套件与GNOME环境相整合，而GNOME也将选择OpenOffice.org作为办公套件，IBM公司则为 GNOME共享了SashXB极速开发环境。同时，GNOME基金会也决定采用Mozilla作为网页浏览器。KDE阵营也毫不示弱，在当年10月份推出万众瞩目的KDE 2.0。KDE2.0堪称当时最庞大的自由软件，除了KDE平台自身外，还包括Koffice办公套件、Kdevelop集成开发环境以及 Konqueror网页浏览器。尽管这些软件都还比较粗糙，但KDE 2.0

已经很好实现了MatthiasEttrich成立KDE项目的目标。也是在这个月，TrollTech公司决定采用GPL公约来发行Qt的免费版本，希望能够以此赢得开发者的支持。这样，Qt实际上就拥有双重授权：如果对应的Linux发行版采用免费非商业性的方式进行发放，那么使用KDE无须向TrollTech交纳授权费用；但如果Linux发行版为盈利性的商业软件，那么使用KDE时必须获得授权。由于TrollTech是商业公司且一直主导着KDE的方向，双许可方式不失为解决开源与盈利矛盾的好办法。TrollTech宣称，双许可制度彻底解决了KDE在GPL公约方面的问题，但RedHat并不喜欢，RedHat不断对GNOME项目提供支持，希望它能够尽快走向成熟，除RedHat之外的其他Linux厂商暂时都站在KDE这一边，但他们同时也在发行版中捆绑了GNOME桌面。

在2001-2002年，火热一时的Linux运动开始陷入低潮期，几乎所有的厂商都发现桌面Linux版本不可能盈利，而易用性的不足也让业界不看好 Linux进入桌面的前途。但在服务器市场，Linux发展势头非常迅猛，直接对UNIX和WindowsServer造成威胁。不过，秉承自由软件理念的开发者们并不理会外界的论调，他们一直将Linux桌面化作为目标，GNOME项目和KDE项目都在这期间获得完善发展。2001年4 月，GNOME1.4发布，它修正了之前版本的Bug，功能也较为完善，但在各方面与KDE依然存在差距；同年8月，KDE发展到2.2版本。2002年 4月，KDE跳跃到3.0版本，它以Qt 3.0为基础，各项功能都颇为完备，具备卓越的使用价值；两个月后，GNOME阵营也推出2.0版本，它基于更完善的GTK2.0图形库。进入到2003 年后，KDE与GNOME进入真正意义上的技术较量。1月份，KDE 3.1推出，而GNOME2.4则在随后的2月份推出，两大平台都努力进行自我完善。也是在这一年，Linux商业界出现一系列重大的并购案：1月份，Novell公司宣布收购德国的SuSE Linux，而SuSELinux是地位仅次于RedHat的全球第二大Linux商业企业；8月，Novell接着将GNOME的母公司Ximian收归旗下。这两起并购案让Novell成为实力与RedHat不相上下的强大Linux企业，而Novell和RedHat就成为能够影响Linux未来的两家企业。在图形环境上，SuSE一向选择KDE，并在KDE身上投入相当多的精力，在被Novell并购后，SuSE的桌面发行版尽管还侧重于KDE，但同样不喜欢Qt授权的Novell已经开始向GNOME迁移。

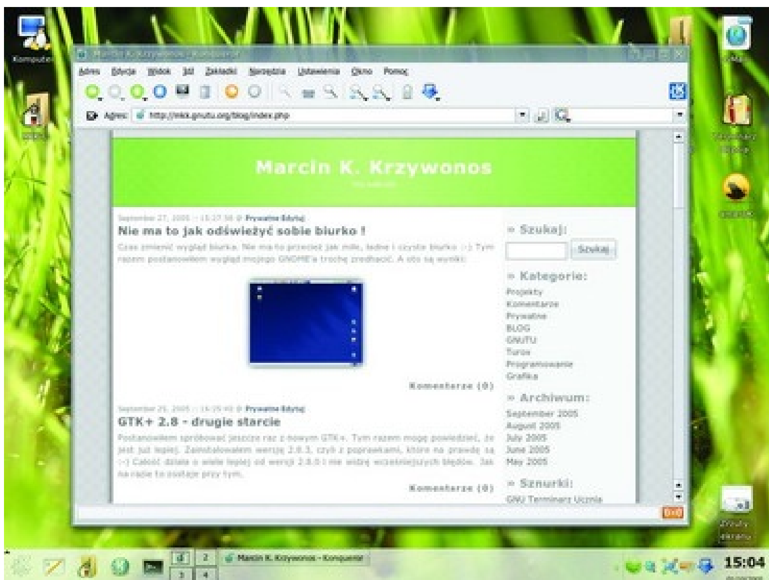
图 **49.5. GTK**库是**GNOME**项目的基础，它完全采用**GPL**授权因此获得广泛支持。



GNOME获得商业公司的支持

进入2004年后，KDE与GNOME依然保持快速发展，KDE阵营分别在2月份和8月份推出3.2、3.3版本，GNOME则在3月和9月推出2.6和 2.8，两者的版本升级步幅旗鼓相当。到3.3版本的KDE已经非常成熟，它拥有包括KOffice、Konqueror浏览器、Kmail套件、KDE 即时消息在内的一大堆应用软件，且多数都达到可用标准，功能上完全不亚于Windows 2000。而GNOME更是在此期间高速发展，GNOME2.8版本的水准完全不逊于KDE3.3，而且此时两者的技术特点非常鲜明：GNOME讲究简单、高效，运行速度比KDE更快；KDE则拥有华丽的界面和丰富的功能，使用习惯也与微软Windows较类似。商业支持方面，RedHat还是 GNOME的铁杆支持者，IBM、SUN、Novell、HP等重量级企业也都选择GNOME，而KDE的主要支持者暂时为SuSE、Mandrake以及中科红旗、共创开源在内的国内发行商。2005年，厚积薄发的GNOME开始全面反超，3月份的2.10、9月份的2.12让GNOME获得近乎脱胎换骨的变化，加之OpenOffice.org 2.0、Firefox1.5等重磅软件的出台让GNOME如虎添翼；KDE方面则分别在3月和11月推出3.4和3.5，其中KDE3.5也逼近完美境地，我们认为它的水平与GNOME2.12不相伯仲。但KDE在商业支持方面每况愈下，Novell在11月宣布旗下所有的商业性发行版将使用GNOME 作为默认桌面（仍会对KDELibraries提供支持），SuSELinux桌面版则会对KDE与GNOME提供同等支持，而社区支持的 OpenSuSE仍将使用KDE体系—但谁都明白GNOME将成为Novell的重心，KDE只是活跃在免费的自由发行版中。

图 **49.6. KDE3.5**可实现半透明和阴影效果，界面华丽、软件丰富。



到这里，我们发现一个颇富戏剧性的结局：致力于商业化的KDE反而失去了重量级商业企业的支持，尽管一些中小规模的Linux企业因技术能力问题将继续支持KDE，但它的商业前途有限。而遵循GPL、完全不以商业化为目的的GNOME反而在该领域大获成功。许多Linux发烧友都不明白为什么优秀的KDE 会受到如此待遇，其实道理非常简单—没有哪一家重量级企业喜欢受制于人，也许KDE的Qt不需要很多授权费，但谁知道TrollTech公司以后会不会漫天要价？既然有免费的GNOME可以选择，那为什么不呢？基于此种理由，RedHat、Novell两家最大的Linux企业和SUN都采用GNOME，而它们对GNOME的鼎力支持也让该项目可拥有足够多的技术保证，为今后的高速发展奠定坚实的基础。需要纠正一个可能的误解，虽然Novell收购了 Ximian，但RedHat并没有受到太大影响，双方对GNOME的贡献都是相互共享的，因

为GNOME以GPL自由版权公约发行，合作即共赢。至于 KDE项目，虽然它失去这些商业巨头的支持，但没有能力转换桌面的中小Linux厂商将继续追随KDE，而且在非商业的社区Linux发行版中，KDE依然有强大的生命力。

图 **49.7. GNOME 1.4**解决了稳定性问题，功能初步完善



虽然在商业方面存在竞争，GNOME与KDE两大阵营的开发者关系并没有变得更糟，相反他们都意识到支持对方的重要性。如果KDE和GNOME无法实现应用程序的共享，那不仅是巨大的资源浪费，而且将导致Linux出现根本上的分裂。事实上，无论是GNOME的开发者还是KDE的开发者，他们都有着共同的目标，就是为Linux开发最好的图形环境，只是因为理念之差而分属不同的阵营。KDE与GNOME的商业竞争对开发者们其实没有任何利益影响（只有TrollTech会受影响），基于共同的目的，KDE与GNOME阵营大约从2003年开始逐渐相互支持对方的程序—只要你在KDE环境中安装GTK 库，便可以运行GNOME的程序，反之亦然。经过两年多的努力，KDE和GNOME都已经实现高度的互操作性，两大平台的程序都是完全共享的，例如你可以在GNOME中运行Konqueror浏览器、Koffice套件，也可以在KDE中运行Evolution和OpenOffice.org，只不过执行本地程序的速度和视觉效果会好一些。在未来一两年内，KDE和GNOME将进行更高等级的融合，但两者大概永远都不会合为一体—GNOME还是GNOME，KDE也还是KDE。或许你觉得这是浪费开发资源而且很可能让用户无从选择，但我们告诉你这就是Linux，它与Windows和 MacOSX有着绝然不同的文化。更何况全球有越来越多自由软件开发者（所以不必担心浪费开发资源），Linux用户的使用偏好也不可能总是相同，保持两个并行发展的图形环境项目没有什么不妥。至于GNOME项目和KDE项目的开发者们，曾经因为理念不同而吵得天翻地覆，但他们现在尽释前嫌，因为所有人都意识到，他们其实彼此需要，团结在一起可以让他们在硬件厂商面前有更大的发言权，从而促使厂商在推出Windows驱动的同时也提供相应的Linux版本，而且彼此可以相互借鉴优秀的设计，确保Linux拥有一个最出色的图形桌面环境。

图 **49.8. GNOME 2.12**保持惯有的简洁和高效



KDE与GNOME走向融合

2006年，GNOME与KDE都站在一个全新的起点，获得商业公司和更多自由程序员支持的GNOME踌躇满志，将超越的目光放在Mac OSX系统。也许你认为WindowsVista的半透明和三维界面将Linux远远抛在后面，那么我们告诉你这是绝对的误解，GNOME目前已经可以实现类似的效果，Novell在前几个月就向外界作过详细的演示。当前的KDE也可支持相当不错的半透明和阴影特效，技术上毫不落后于GNOME。现在，GNOME项目朝向革命性的3.0版本迈进，KDE则致力于开发同样有重大技术变革的4.0，这两个成果大概在2007年可进入现实，届时Linux 系统将具备更卓越的可用性。也就是说，Linux桌面应用的全面铺开指日可待，而除了开发者和厂商的努力外，如何向企业和个人用户推广以及提供培训将是厂商要考虑的主要问题，我们今天恰好站在这样的一道门槛上。

第 50 章 Vim Emacs

第 51 章 年代纪

第 52 章 我的选择

目录

类别	首选	备选
发行版	archlinux	freeBSD gentoo debian
Shell	zsh	bash
终端	rxvt-unicode	tilda screen
编辑器	Emacs	Vim jed
文档写作	docbook	reST muse
版本控制	git	svn
加密	PGP	truecrypt
窗口管理器	awesome openbox	xfce4
浏览器	Firefox	liferea
桌面小程序	conky	
输入法	Fcitx	scim
下载工具	aria2 lftp	wget
播放器	mplayer w32codecs	
图片浏览	gqview	
电子阅读	evince	
词典	sdcv	
文件管理	tar gzip p7zip	squeeze unzip unrar
日程管理	osmo	
网络服务	sshd pureftpd	lighttpd php mysql phpmyadmin
编程	C python	ruby lisp
其它	sudo netcfg	

第 53 章 补遗